Proceedings of the Ninth Canadian Conference on Computational Geometry

Queen's University Kingston, Ontario August 11-14, 1997

Foreword

The Ninth Canadian Conference on Computational Geometry was held from August 11-14, 1997, at Queen's University in Kingston, Ontario. This annual conference attracts researchers in computational geometry from around the world to Canada, for an open exchange of new ideas and results. The goal of the conference is to encourage research in a broad area of topics, and to promote collaboration. This volume contains the abstracts of 46 contributed papers, as well as four invited talks by J. Akiyama (Tokai University), W. Haken (University of Illinois), E. Fiume (University of Toronto), and J. Mitchell (SUNY Stony Brook).

The organizing committee thanks those who submitted papers to the conference, the invited speakers, the members of the program committee, and especially all those who attended the conference. We gratefully acknowledge the financial support from the Information Technology Research Centre of Ontario, the Office of the Vice Principal (Research) of Queen's University, the Office of the Dean of the Faculty of Arts and Science, Queen's University, and the Department of Computing and Information Science, Queen's University. Special thanks to Nancy Barker for her extraordinary administrative assistance and to Linda Moulton for her assistance in preparing this proceedings.

David Rappaport

Avant-Propos

La neuvième conférence canadienne sur la géométrie algorithmique a eu lieu du 11 au 14 août 1997, à l'Université Queen's à Kingston, en Ontario. Cette conférence annuelle réunit au Canada des chercheurs en géométrie algorithmique du monde entier, pour un échange d'idées et de résultats. Le but de ces rencontres est d'encourager la recherche sur des sujets d'une grande diversité et de promouvoir la collaboration. Ce volume contient les textes de 46 articles communiqués à la conférence, ainsi que les résumés de 4 articles présentés sur invitation par J. Akiyama (Tokai University), W. Haken (University of Illinois), E. Fiume (University of Toronto), et J. Mitchell (SUNY Stony Brook).

Le comité organisateur remercie les auteurs des communications, les conférenciers invités, les membres du comité de programme et surtout les participants. Nous désirons exprimer notre reconnaissance à l'Information Technology Research Centre of Ontario, à l'Office of the Vice Principal (Research), à l'Office of the Dean of the Faculty of Arts and Science, et au Department of Computing and Information Science, de l'Université Queen's, de leur support financier. Nous tenons également à offrir notre gratitude à Nancy Barker pour l'extraordinaire assistance administrative qu'elle nous a apportée et à Linda Moulton pour son aide à la préparation de ces comptes-rendus.

David Rappaport

The Ninth Canadian Conference on Computational Geometry

August 11-14, 1997
Queen's University
Kingston, ON, CANADA

Table of Contents

Session TUE1

Session TUE2

Session TUE3 Invited Talk

Session TUE4

Session WED1

Session WED2

Session WED3 Invited Talk

Session WED4

Session WED5

Session THU1 Invited Talk

Session THU2

Session THU3

Vertex π-Lights for Monotone Mountains

Joseph O'Rourke *

Abstract

It is established that $\lceil t/2 \rceil = \lceil n/2 \rceil - 1$ vertex π-lights suffice to cover a monotone mountain polygon of $t = n-2$ triangles. A monotone mountain is a monotone polygon one of whose chains is a single segment, and a vertex π-light is a floodlight of aperture π whose apex is a vertex.

Keywords. art gallery theorems, floodlights, monotone polygons.

1 Introduction

It was established in [ECOUX95] that for any $\alpha < \pi$, there is a polygon that cannot be illuminated with an α-floodlight at each vertex. An *α-floodlight* (or *α-light*) is a light with aperture no more than α. A *vertex α-light* is one whose apex is placed at a vertex, aiming a cone of light of up to α into the polygon. Each vertex may be assigned at most one light. The result of [ECOUX95] is then that n vertex α-lights do not always suffice when $\alpha < \pi$. Let a polygon P have t triangles in any triangulation, $t = n - 2$; we will phrase bounds in terms of t. For $\alpha = \pi$, an easy argument shows that t vertex π-lights always suffice: place a light at an ear tip, cut off the ear, and recurse. This raises the question of finding a better upper bound. Urritia phrased the problem this way [Urr97]: is there a $c < 1$ such that cn vertex π-lights always suffice? The largest lower bound is $c = \frac{3}{5}$ via an example of F. Santos.

In this paper we pursue this question, but only in special cases. In particular, we show that $c = \frac{1}{2}$ for spirals and, more interestingly, for monotone mountains. A *monotone mountain* is a monotone polygon one of whose chains is a single segment. More precisely, a *monotone chain* is a polygonal chain whose intersection with any vertical line is at most one point. A monotone mountain consists of one monotone chain, whose extreme (left and right) vertices are connected by a single segment. Note this *base edge* need not be horizontal.[1] Fig. 5 shows a monotone mountain with base edge xy.

Although this is a severely restricted class of polygons, it deserves attention for three reasons: the examples establishing the results of [ECOUX95] (and [OX94]) are "nearly" monotone mountains; the problem is already not completely trivial for monotone mountains; and there is some reason to hope similar techniques will apply to the unrestricted problem.

We start with a result on spiral polygons, where the problem is trivial.

2 Spiral Polygons

A *spiral polygon* consists of two joined polygonal chains: a chain of reflex vertices, and a chain of convex vertices.

Theorem 1 *A spiral polygon S of $t = n-2$ triangles may be covered by $\lceil t/2 \rceil = \lceil n/2 \rceil - 1$ vertex π-lights; some spirals require this many.*

Proof: If S has no reflex vertices, S is convex and can be covered with one vertex π-light at any vertex. So assume S has at least one reflex vertex.

Let x, y, and z be three consecutive vertices of S, with x reflex, y convex, and z convex. Such a triple always exists, because any polygon has at least three convex vertices. The segment xz must be an internal diagonal of the polygon. Therefore at least two triangles are incident to z in any triangulation of S. Placing a light at z, as shown in Fig. 1, therefore covers at least two triangles; because z is convex, the light covers the entire angle at z. Removing the covered

*Department of Computer Science, Smith College, Northampton, MA 01063, USA. orourke@cs.smith.edu. Supported by NSF grant CCR-9421670.

[1] This definition differs in this respect from that introduced in [OX94], which demanded a horizontal base edge.

triangles leaves a smaller spiral polygon. Repeating this process covers S with at most $\lceil t/2 \rceil$ lights.

Generalizing the polygon shown in Fig. 1 establishes that the bound is tight. □

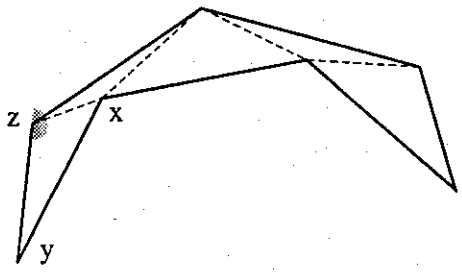


Figure 1: Placing a π-light at z covers at least two triangles. The light is shown as a full π-light, although only the angle interior to the polygon is relevant.

Notice that the procedure implied by this proof places lights only on convex vertices. One reason spiral polygons are so easy is that lights never need be placed on reflex vertices, and so the potentially difficult decision of how to orient a π-light at a reflex vertex need not be confronted.

3 Non-Locality

Monotone mountains are more difficult than spirals for two reasons: reflex vertices cannot be avoided, and the decision of how of orient a light at reflex vertex cannot be made locally. Many art gallery theorems can be proved inductively as follows: cut off a small piece, illuminate that piece, and recurse on the remainder [O'R87]. The reason this paradigm works is that decisions can be made locally: what happens in the small piece is independent of the shape of the remainder of the polygon.

This is not the case with the vertex π-light problem, even for monotone mountains. Consider the polygon shown in Fig. 2, and imagine trying to decide whether to shine the light at z left or right, basing the decision only on the portion of the polygon to the left of z. One can see that no $c < 1$ can be achieved without looking at the structure of the right portion: if the "wrong" decision is made at z (as illustrated), then an arbitrarily large fraction of all remaining vertices will need lights. Although the decision is obvious in this case, as it can be based on the number of triangles incident to z, the effect might be more subtle.

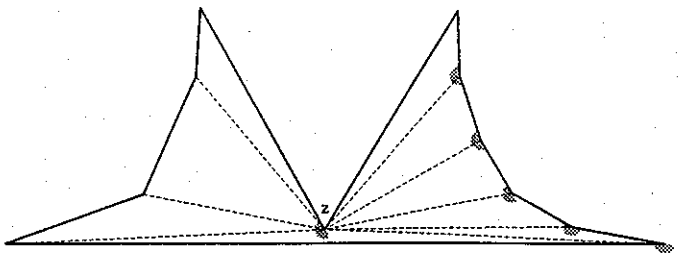


Figure 2: A wrong orienting decision at z can lead to suboptimal coverage.

4 Worst Case

It is clear that if the number of triangles incident to z in Fig. 2 from the left is k and from right is also k, then a lower bound of $c = \frac{1}{2}$ is attained: $t = 2k + 1$, and $k + 1 = \lceil t/2 \rceil$ lights are necessary, one at z and k on the opposite reflex chain. The same bound is acheived by the shape shown in Fig. 3. In this polygon, the extension of v_1v_2 meets v_5v_6; the extension of v_2v_3 meets v_4v_5; and so on.

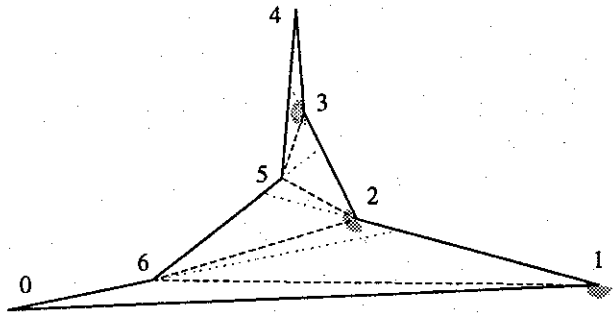


Figure 3: $\lceil t/2 \rceil$ lights are necessary: $t = 5$ and $\lceil 5/2 \rceil = 3$ are needed.

We prove this simple fact for later reference:

Lemma 1 *The generalization of the polygon M in Fig. 3 requires $\lceil t/2 \rceil = \lceil n/2 \rceil - 1$ vertex π-lights.*

Proof: Each vertex on the left chain can only see two vertices on the right chain, and vice versa: v_5 can see v_2 and v_3, because the extensions of v_1v_2 and v_2v_3 straddle v_5; etc. Thus at most (in fact exactly) three triangles are incident to v in a triangulation of M. A π-light at v can only fully cover two of these three triangles, because v is reflex. So each light covers at most two triangles, and $\lceil t/2 \rceil$ are needed overall. □

5 Duality

One way to view the phenomenon illustrated in Fig. 2 is as follows: the polygon naturally partitions into two monotone mountain subpolygons at z. If at light is placed at z and aimed left, then in the right subpolygon, placing a light at z is forbidden (as that would place two lights at one vertex). Moreover, that example shows that a (sub)polygon with one vertex forbidden a light could in fact require one light per triangle.

However, there is an interesting "duality" at play here, in the following sense: if a polygon with one forbidden vertex requires many lights, then placing a light at the forbidden vertex permits it to be covered with few lights. In other words, there is no polygon structure that is both bad with a forbidden vertex and bad without that vertex forbidden.

If M is a monotone mountain with extreme left and right vertices x and y, let $L_{10}(M)$ be the number of vertex π-lights needed to cover M when vertex x is assigned a light and y is forbidden to have a light; and let $L_{01}(M)$ be the number needed when y is assigned a light and x is forbidden. Note that, in these definitions, not only is one vertex forbidden a light, but the other extreme vertex must be assigned a light. The precise statement of duality is captured in the following lemma:

Lemma 2 *For any monotone mountain M of t triangles, $L_{10}(M) + L_{01}(M) \leq t + 1$.*

The generalization of Fig. 4 establishes that the sum is sometimes as large as $t+1$: here $L_{10}(M) = 1$ (v_0 assigned) and $L_{01}(M) = t$ (v_0 forbidden, as illustrated).

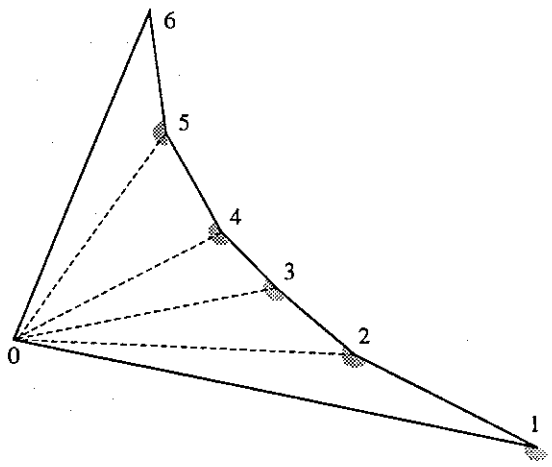


Figure 4: Duality: $L_{10}(M) + L_{01}(M) = t + 1$.

Lemma 2 is the key to the main theorem in the next section. We now prove it via induction.

Proof: Let M be a monotone mountain of t triangles. The induction hypothesis is that $L_{10}(M') + L_{01}(M') \leq t' + 1$ for any monotone mountain M' of $t' < t$ triangles. The base case is a single triangle T, $t = 1$, when $L_{10}(T) = L_{01}(T) = 1$, and so $L_{10}(T) + L_{01}(T) = 2 = t + 1$.

Let the base edge of M be xy, and let z be the vertex first encountered by sweeping the line containing xy vertically; see Fig. 5. It must be the case that both xz and yz are internal diagonals. This provides a natural partition of M into three pieces: $\triangle xyz$, a subpolygon A sharing diagonal xz, and a subpolygon B sharing diagonal yz. Note that it may well be that either A or B is the empty polygon $\emptyset$; if both are empty, $t = 1$ and we fall into the base case of the induction.

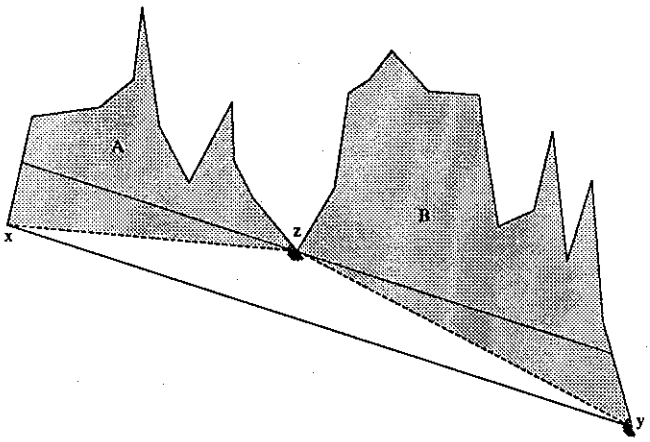


Figure 5: Induction partition of M into A, B, and $\triangle xyz$.

It is clear that A and B are monotone mountains. In particular, the angle at z in A is convex, as is the angle at z in B: for the monotone chain enters z from the left and leaves it from the right (Fig. 6), as do the diagonals xz and zy respectively.

We prove the lemma in two cases.

Case 1: Neither A nor B is empty.

We compute a bound on $L_{10}(M)$, which places a light at x but forbids a light at y. Because the angle at x in M is convex, the light at x covers $\triangle xyz$. This light also serves as a light at x in A. It makes sense in this situation to place a light at z and aim it into B. Doing this gives us an upper bound on $L_{10}(M)$, upper because this sensible light placement and orientation at z might not optimal. This strategy yields

$$L_{10}(M) \leq L_{10}(A) + L_{10}(B). \qquad (1)$$

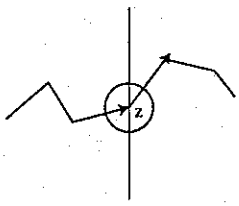


Figure 6: The monotone chain enters each vertex from the left halfplane and leaves in the right halfplane.

Analogous reasoning (again the light at y (illustrated in Fig. 5) covers $\triangle xyz$) yields

$$L_{01}(M) \leq L_{01}(A) + L_{01}(B)\,. \qquad (2)$$

Adding Eqs. 1 and 2 yields

$$L_{10}(M)+L_{01}(M) \leq [L_{10}(A)+L_{01}(A)]+[L_{10}(B)+L_{01}(B)]\,.$$

Suppose A contains a triangles and B contains b triangles, so that $t = a + b + 1$. Then applying the induction hypothesis to each yields

$$\begin{aligned} L_{10}(M) + L_{01}(M) &\leq [a+1] + [b+1] \\ L_{10}(M) + L_{01}(M) &\leq t+1\,. \end{aligned}$$

This is the claim to be proved.

It only remains to handle the case where one of A or B is empty.

Case 2: $A = \emptyset$ but B is not empty.

This case is illustrated in Fig. 7; the case with $B = \emptyset$ is symmetric and need not be considered. If a light is placed at x, it serves to cover $\triangle xyz$, and the reasoning is just as before:

$$L_{10}(M) \leq 1 + L_{10}(B)\,.$$

If a light is placed at y, then it covers $\triangle xyz$ (as illustrated in Fig. 7), and there is no need to an additional light to cover the empty A:

$$L_{01}(M) \leq L_{01}(B)\,.$$

Adding yields

$$\begin{aligned} L_{10}(M) + L_{01}(M) &\leq 1 + [L_{10}(B) + L_{01}(B)] \\ L_{10}(M) + L_{01}(M) &\leq 1 + [b+1] \\ L_{10}(M) + L_{01}(M) &\leq t+1\,. \end{aligned}$$

□

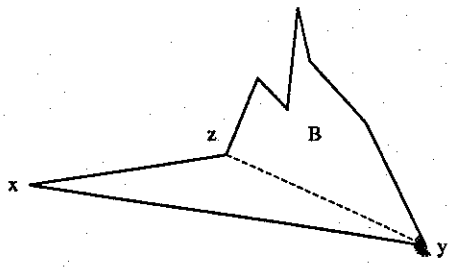


Figure 7: $A = \emptyset$.

6 Main Result

With Lemma 2 in hand, the final step is easy:

Theorem 2 *A monotone mountain polygon M of $t = n-2$ triangles may be covered by $\lceil t/2 \rceil = \lceil n/2 \rceil - 1$ vertex π-lights; some monotone mountains require this many.*

Proof: We know from Lemma 2 that

$$L_{10}(M) + L_{01}(M) \leq t+1\,.$$

Let

$$L(M) = \min\{L_{10}(M), L_{01}(M)\}\,.$$

By the pigeonhole principle,

$$L(M) \leq \lfloor (t+1)/2 \rfloor = \lceil t/2 \rceil\,.$$

Lemma 1 established that this bound can be attained (Fig. 3). □

The proofs of Lemma 2 and Theorem 2 imply a simple algorithm: compute a bound on $L_{10}(M)$ by placing lights at the left corners of A and of B and recursing, and compute a bound on $L_{01}(M)$ similarly. Use the light placement of whichever is smaller. The algorithm is easily seen to be $O(n \log n)$: spend linear time finding z, and recursively process the pieces. This leads to the familiar divide-and-conquer recurrence.

An example is shown in Fig. 8. Here M has $t = 14$ triangles, and $L_{10}(M) + L_{01}(M) \leq 5 + 10 = t + 1$. This example illustrates a number of features of the light placements implied by the bound computation on L_{10} and L_{01}:

1. Every vertex that is not a local maximum is assigned a light in either the L_{10} or L_{01} computation. (Some vertices are assigned a light in both.)

2. All the lights in the L_{10} placement aim to the right; and all those in the L_{01} placement aim to the left.

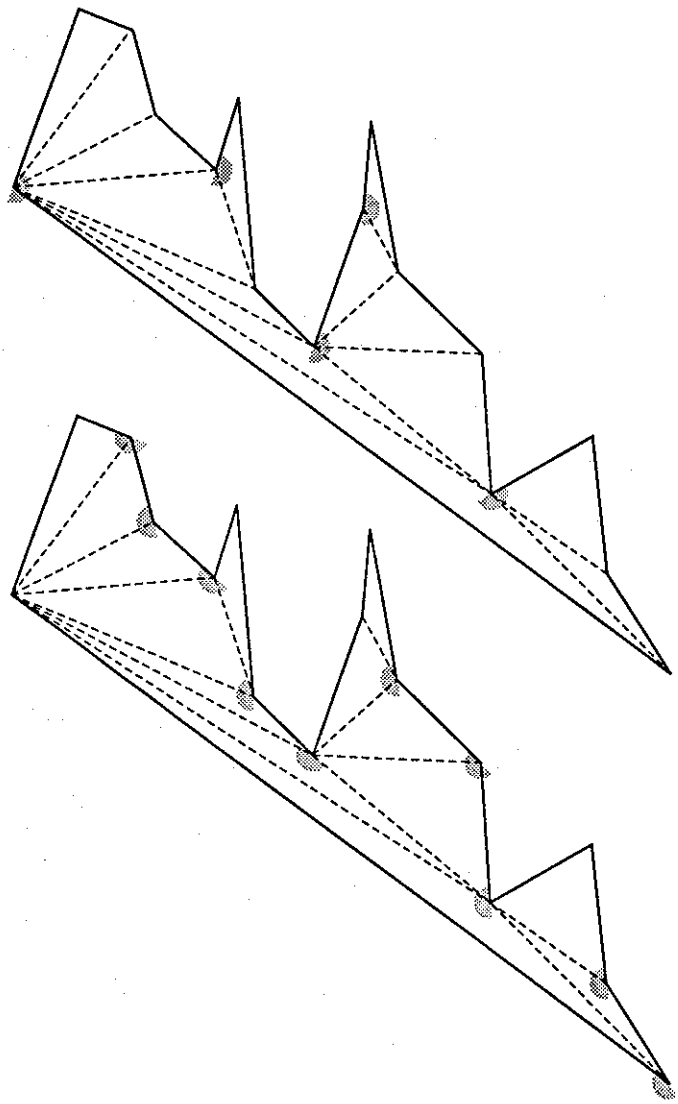


Figure 8: Example: $t = 14$, $L_{10} = 5$ (top), $L_{01} = 10$ (bottom).

3. The sum $L_{10}(M) + L_{01}(M)$ achieved is always exactly $t + 1$, because blindly following the procedure places lights even if they might not be needed (e.g., when M is convex).

4. Lights at reflex vertices are turned either fully counterclockwise (in L_{10}) or clockwise (in L_{10}): intermediate positions are never needed.

7 Discussion

Many of the features present in monotone mountains hold for the problem for general simple polygons as well: for example, non-locality. For other features, it remains unclear: for example, whether every light may be fully turned (observation 4 above). In any case, I believe that a version of the duality described in Lemma 2 holds and will be a key to solving Urrutia's problem. I conjecture $c = \frac{2}{3}$ is achievable.

References

[ECOUX95] V. Estivill-Castro, J. O'Rourke, J. Urrutia, and D. Xu. Illumination of polygons with vertex floodlights. *Inform. Process. Lett.*, 56:9–13, 1995.

[O'R87] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, New York, NY, 1987.

[OX94] J. O'Rourke and D. Xu. Illumination of polygons with 90° vertex lights. In *Snapshots of Computational and Discrete Geometry*, volume 3, pages 108–117. School Comput. Sci., McGill Univ., Montreal, PQ, July 1994. Technical Report SOCS-94.50.

[Urr97] J. Urrutia. Art gallery and illumination problems. In J.-R. Sack and J. Urrutia, editors, *Handbook on Computational Geometry.* North-Holland, 1997. To appear.

Constructing Piecewise Linear Homeomorphisms of Polygons with Holes

Mark Babikov Diane L. Souvaine* Rephael Wenger†

Abstract

Let P and Q be two homeomorphic polygons with holes with vertex sets $\{p_1, \ldots, p_n\}$ and $\{q_1, \ldots, q_n\}$, respectively, such that there is a homeomorphism from P to Q mapping p_i to q_i. We show how to construct in $O(n^2)$ time a piecewise linear homeomorphism from P to Q mapping p_i to q_i using $O(n^2)$ triangles. Equivalently, we construct in $O(n^2)$ time isomorphic triangulations of P and Q identifying p_i with q_i using $O(n^2)$ triangles.

1 Introduction

Let P and Q be two polygons in $\mathbb{R}^2$, possibly with holes. Two triangulations $\mathcal{T}_P$ and $\mathcal{T}_Q$ of P and Q, respectively, are *isomorphic* if there is a one to one, onto mapping f from the vertices of $\mathcal{T}_P$ to the vertices of $\mathcal{T}_Q$ such that (p, p', p'') is a triangle of $\mathcal{T}_P$ if and only if $(f(p), f(p'), f(p''))$ is a triangle of $\mathcal{T}_Q$. These triangulations may contain many additional boundary and interior vertices beyond the original vertices of P and Q. Our goal is to construct isomorphic triangulations of P and Q, assuming they exist, using as few triangles as possible.

More specifically, let $\{p_1, \ldots, p_n\}$ and $\{q_1, \ldots, q_n\}$ be the vertices of P and Q, respectively. We would like to construct isomorphic triangulations of P and Q where vertex p_i is mapped to vertex q_i, assuming such triangulations exist. Such triangulations may require additional vertices.

Our motivation for constructing isomorphic triangulations comes from constructing piecewise linear homeomorphisms. A *homeomorphism* from P to Q is a one to one, onto, continuous map with continuous inverse from polygon P and its interior to polygon Q and its interior. A homeomorphism h from P to Q is *piecewise linear* if P has a triangulation $\mathcal{T}_P$ such that the homeomorphism is linear on each triangle.

Isomorphic triangulations $\mathcal{T}_P$ and $\mathcal{T}_Q$ of P and Q induce a piecewise linear homeomorphism h from P to Q which linearly maps each triangle in $\mathcal{T}_P$ to the corresponding triangle in $\mathcal{T}_Q$ as follows. For each vertex v of $\mathcal{T}_P$, let $h(v)$ be the corresponding vertex of $\mathcal{T}_Q$. For every point $p \in P$, lying in triangle (v, v', v'') of $\mathcal{T}_P$, express p in barycentric coordinates as $\alpha v + \alpha' v' + \alpha'' v''$ where $\alpha, \alpha', \alpha'' \in \mathbb{R}$ and let $h(p)$ equal $\alpha h(v) + \alpha' h(v') + \alpha'' h(v'')$. A little checking is needed to be sure that h is well-defined.

Conversely, a piecewise linear homeomorphism h from P to Q induces isomorphic triangulations of P and Q. If $\mathcal{T}_P$ is a triangulation of P and h is linear on each triangle of $\mathcal{T}_P$, then the image of each triangle in $\mathcal{T}_P$ is a triangle of Q and these triangles form a triangulation of Q.

Let P and Q be two homeomorphic polygons with holes with vertex sets $\{p_1, \ldots, p_n\}$ and $\{q_1, \ldots, q_n\}$, respectively. In this paper we present an algorithm for constructing isomorphic triangulations and piecewise linear homeomorphisms between P and Q mapping p_i to q_i, assuming such triangulations and homeomorphisms exist. The algorithm runs in $O(n^2)$ time and the construction uses $O(n^2)$ triangles.

Aronov, Seidel and Souvaine in [1] showed how to construct isomorphic triangulations and piecewise linear homeomorphisms of simple polygons in $O(n^2)$ time using $O(n^2)$ triangles where n was the number of vertices of each polygon. They also showed that this number of triangles is sometimes required. Their algorithm allows one to prespecify which polygon vertices must be identified.

Kranakis and Urrutia in [5] gave an algorithm for constructing isomorphic triangulations of simple polygons P and Q which ran in $O(n + r_P r_Q \log n)$

*Rutgers University, Piscataway, NJ 08855, U.S.A. (dls@cs.rutgers.edu). Supported in part by NSF grants CCR-91-04732 and NCR-95-27163.

†Ohio State University, Columbus, OH 43210, U.S.A. (wenger@cis.ohio-state.edu). Supported by NSA grant MDA904-97-1-0019.

time using $O(n + r_P r_Q)$ triangles where n is the number of vertices of each polygon and r_P and r_Q are the number of reflex vertices of polygons P and Q, respectively. They also gave a different algorithm which runs in $O(n + (r_P + r_Q)^2)$ time and uses $O(n + (r_P + r_Q)^2)$ triangles.

Saalfeld in [8] gave an algorithm for constructing isomorphic triangulations and piecewise linear homeomorphisms between two rectangles P and Q under the constraint that certain interior points $p_i \in P$ must map to corresponding interior points $q_i \in Q$. Both the running time and the number of triangles was potentially exponential. Souvaine and Wenger in [9] gave an $O(n^2)$ algorithm using $O(n^2)$ triangles for the same problem. Pach, Shahrokhi and Szegedy in [7] proved that $\Omega(n^2)$ triangles are sometimes required.

Interior points of a rectangle or a polygon can be modelled as vertices of "degenerate" holes consisting of single points. Thus the algorithm in this paper for constructing isomorphic triangulations and piecewise linear homeomorphisms between polygons with holes applies equally well to the problem when certain interior points must be matched as in [8] and [9].

We do not know whether isomorphic triangulations using the fewest number of triangles can be constructed in polynomial time or whether the problem is NP-complete. Gupta and Wenger in [3] give an approximation algorithm for constructing isomorphic triangulations between simple polygons which runs in $O(M \log n + n \log^2 n)$ time and used $O(M \log n + n \log^2 n)$ triangles, where M is the number of triangles in the optimal solution. They slightly improved this algorithm in [2] to run in $O(M \log n)$ time using $O(M \log n)$ triangles. We know of no similar approximation algorithm for polygons with holes.

2 Isomorphic Triangulations

Let P be a polygon with h holes. The boundary of P is composed of $h + 1$ simple closed curves, one for each hole and one for the outer boundary of P. From the standpoint of topology, the outer boundary of P is no different from the boundary of any of the holes. There is a homeomorphism from P to itself which maps the boundary of P to the boundary of any of the holes in P.

We start by describing how to connect the simple closed curves on the boundary of P using $O(n)$ line segments so that these closed curves form the vertices of a path. This path will guide the construction of the isomorphic triangulation.

Lemma 1 *Let P be a polygon with holes on n vertices. In $O(n)$ time and using at most $6n$ line segments, the simple closed curves bounding P can be connected by polygonal curves so that they form the vertices of a path. Moreover, any simple closed curve bounding P can be chosen to represent the first vertex in the path.*

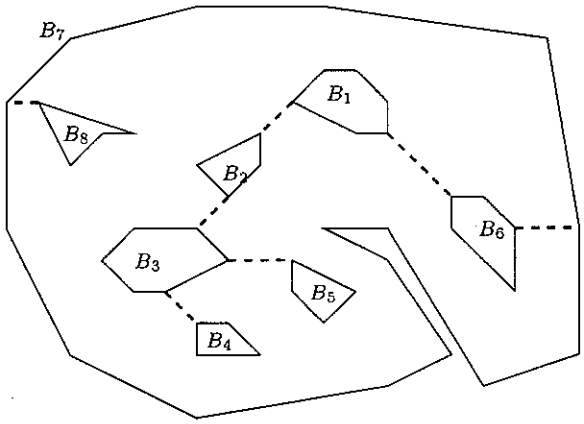


Figure 1: Γ representing the tree G_2.

Outline of proof: Triangulate P and let D_1 be the set of triangulation edges. Let G_1 be the multigraph whose vertices are the simple closed curves on the boundary of P and whose edges are the diagonals of the triangulation. Note that graph G_1 is connect. Let G_2 be a spanning subtree of G_1 and D_2 be the diagonals of P which represent the edges of G_2. Let Γ be the union of all the diagonals in D_2 and the boundary of P. (See Figure 1.) Note that Γ is connected and consists of at most $2n$ line segments.

Pick any simple closed curve on the boundary of P and label it B_1. Starting at any vertex of B_1, walk around the boundary of Γ, keeping the contour of Γ on the left. Label the simple closed curves $B_2, \ldots, B_k$ in the order they are first visited (similar to visiting nodes of G_1 in preorder.) Note that a simple closed curve may be passed multiple times but receives just one label.

We connect the simple closed curves bounding P in the order $B_1, \ldots, B_k$ as follows. If some diagonal in D_2 connects B_1 to B_2, then use that same diagonal to connect to B_i and B_{i+1} in the path. Otherwise, connect B_i to B_{i+1} by a polygonal curve which traces along the contour of Γ, keeping that contour on the left. (See Figure 2.)

By following the contours, each edge in Γ is used or traced at most twice and so the connecting polygonal curves use at most $4n$ line segments. □

We need a variation on a theorem by Kranakis and Urrutia from [5]. We first state and prove a slightly

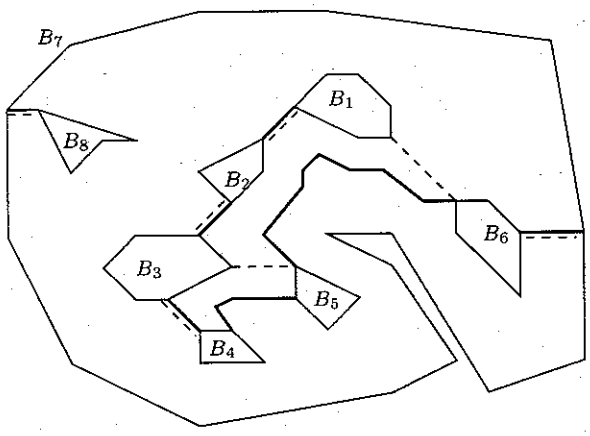


Figure 2: Path connecting holes.

tighter version of this theorem. A *reflex* vertex is a vertex whose interior angle is strictly greater than 180°.

Lemma 2 *Isomorphic triangulations of simple polygons $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_n\}$ matching p_i with q_i can be constructed in $O(n + r_P r_Q)$ time using $O(n + r_P r_Q)$ triangles where r_P and r_Q are the number of reflex vertices of P and Q, respectively. Moreover, the boundary of these isomorphic triangulations contains no new vertices other than $\{p_1, \ldots, p_n\}$ and $\{q_1, \ldots, q_n\}$.*

Proof: The algorithm we present is similar to the one in [5] except that we use the Hertel-Mehlhorn algorithm [4, 6] for convex partitioning instead of ray shooting. Triangulate each polygon. For each reflex vertex choose the one or two diagonals which break the angle at the vertex into angles less than 180°. The chosen diagonals partition each polygon into convex pieces.

Let $C = \{c_1, \ldots, c_n\}$ be a convex polygon on n vertices. Partition C with diagonals corresponding to the chosen diagonals in P and then overlay them with diagonals corresponding to the chosen diagonals in Q. Triangulate each convex cell in the overlay and construct corresponding triangulations of P and Q.

At most $2r_P$ and $2r_Q$ diagonals are chosen from P and Q, respectively. The overlay of the corresponding diagonals in C has at most $4r_P r_Q$ intersection points and so the final triangulations have $O(r_P r_Q)$ triangles. □

We actually need the following corollary to lemma 2.

Corollary 1 *Let $\{s_1, \ldots, s_m\}$ and $\{t_1, \ldots, t_m\}$ be sets of m points on the boundary of simple polygons P and Q, respectively, which include the vertex sets of P and Q and are listed in the order they appear on the boundary. Isomorphic triangulations of P and Q matching s_i with t_i can be constructed in $O(m + n^2)$ time using $O(m + n^2)$ triangles where n is the total number of vertices of P and Q. Moreover, the boundary of these isomorphic triangulations contains no new vertices other than $\{s_1, \ldots, s_m\}$ and $\{t_1, \ldots, t_m\}$.*

Proof: Draw a slightly reduced version P' of P within P where each s_i which is not a vertex of P is replaced by a convex vertex in P', i.e., a vertex whose interior angle is less than 180°. Do the same for Q constructing a polygon Q' inside Q. Triangulate the region between P and P' by adding an edge between each $s_i \in P$ and the corresponding vertex in P' and between s_i and the vertex corresponding to s_{i+1} in P'. Construct a similar triangulation of the region between Q and Q'. Construct isomorphic triangulations of P' and Q' by applying Lemma 2. Since only reflex vertices of P and Q generate reflex vertices of P' and Q', the size of the triangulations are $O(m + n^2)$. □

Two polygons P and Q have the same number of holes if and only if they are *homeomorphic*, i.e., there exists some homeomorphism from P to Q. Equivalently, they have the same number of holes if and only if P and Q have isomorphic triangulations.

If $\{p_1, \ldots, p_n\}$ and $\{q_1, \ldots, q_n\}$ are the vertex sets of P and Q, respectively, then there is a homeomorphism mapping p_i to q_i if and only if P and Q have isomorphic triangulations where p_i is matched to q_i. If there is such a homeomorphism, then two vertices on the closed curve on the boundary of P must correspond to vertices on a corresponding closed curve on the boundary of Q. Note, however, that it is possible for one of these closed curves to bound a hole while the other forms the outer boundary of a polygon.

Theorem 1 *Let P and Q be two homeomorphic labelled polygons with holes with vertex sets $\{p_1, \ldots, p_n\}$ and $\{q_1, \ldots, q_n\}$ such that there is a homeomorphism from P to Q mapping p_i to q_i. Isomorphic triangulations and piecewise linear homeomorphisms of P and Q matching p_i with q_i can be constructed in $O(n^2)$ using $O(n^2)$ triangles.*

Outline of proof: By Lemma 1, use at most $4n$ line segments to connect the simple closed curves bounding P by polygonal curves so that they form the vertices of a path. Let Φ^P be the union of the simple closed curves bounding P and the polygonal curves connecting them. Label the simple closed

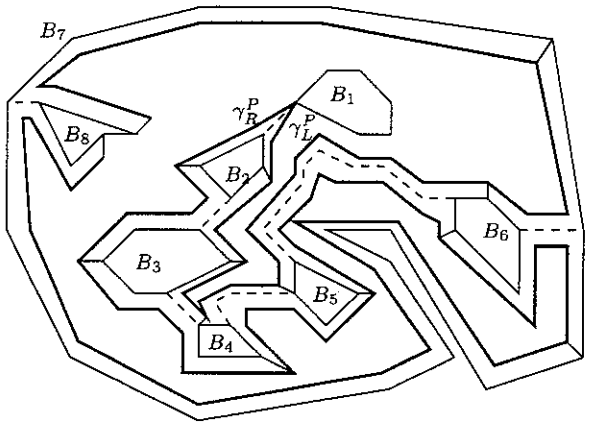


Figure 3: Partition of P by γ_L^P, γ_R^P and $\{\sigma_i^P\}$.

curves bounding P in consecutive order along the path $B_1^P, B_2^P, \ldots, B_k^P$. Starting at B_1^P, draw two polygonal paths γ_L^P and γ_R^P on the left and right of Φ^P and following the contour of Φ^P. For each i from 2 to k, draw line segments $\sigma_{L,i}^P$ and $\sigma_{R,i}^Q$ connecting vertices of B_i^P to vertices of γ_L^P and γ_R^P, respectively. (See Figure 3.) There may be many choices for $\sigma_{L,i}^P$ and $\sigma_{R,i}^Q$.

Let $B_1^Q, B_2^Q, \ldots, B_k^Q$ be the simple closed curves bounding Q where B_i^Q corresponds to the closed curve B_i^P in P. Again applying Lemma 1, connect the simple closed curves bounding Q so that they form the vertices of a path starting at B_1^Q. Note that these holes will not necessarily be connected in consecutive order. Let Φ^Q be the union of the simple closed curves bounding Q and the polygonal curves connecting them. Pick the vertex of B_1^Q corresponding to the common endpoint of γ_L^P and γ_L^Q onn B_1^P and draw two suitably short line segments γ_L^Q and γ_R^Q from this vertex. These line segments will be the images of γ_L^P and γ_R^P. For each vertex of γ_L^P or γ_R^P add a corresponding point on γ_L^Q or γ_R^Q, respectively.

In the interior of P, draw $2k$ polygonal contour lines following the contour of Φ^Q. If δ is the minimum distance between any two non-adjacent edges of Φ^Q, then these contour lines can be spaced a distance $\delta/(8k)$ apart. Number the contour lines 1 to $2k$ starting at the closest one to Φ^Q.

For i from 2 to k, draw polygonal curves $\sigma_{R,i}^Q$ and $\sigma_{L,i}^Q$ from γ_R^Q and γ_L^Q to B_i^P as follows. Start at the point on γ_R^Q corresponding to the endpoint of $\sigma_{R,i}^P$ and draw a polygonal curve in the $(k+1-i)$'th contour line around Φ^Q, keeping Φ^Q on the right. Whenever encountering a boundary curve B_j^Q where $j < i$ cross over the polygonal curve connecting B_{j-1}^Q to B_j^Q, trace along the i'th contour line with B_j^Q to its left, and then cross back over the curve connecting B_j^Q and B_{j+1}^Q and continue. Connect this curve to the point on B_i^Q corresponding to the endpoint of $\sigma_{R,i}^P$. Label this polygonal curve $\sigma_{R,i}^Q$.

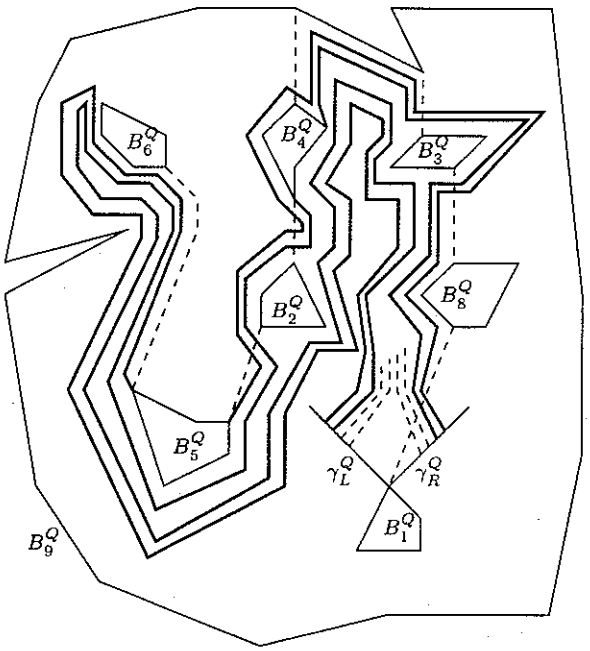


Figure 4: Polygonal curves $\sigma_{R,4}^Q$, $\sigma_{L,4}^Q$, $\sigma_{R,5}^Q$ and $\sigma_{L,5}^Q$.

From the point on B_i^Q corresponding to the endpoint of $\sigma_{L,i}^P$ continue with a polygonal curve in the $(k+1-i)$'th contour line around Φ^Q, keeping Φ^Q on the right. As before, whenever encountering a boundary curve B_j^Q where $j < i$ cross over the polygonal curve connecting B_{j-1}^Q to B_j^Q, trace along the i'th contour line with B_j^Q to its left, and then cross back over the curve connecting B_j^Q and B_{j+1}^Q and continue. When the last closed curve in the path is reached and passed, either on the left or the right, cross to the $(k+i)$'th contour line and return to γ_L^Q by a polygonal curve which keeps Φ^Q on the left. Label this polygonal curve $\sigma_{L,i}^Q$. (See Figure 4.) For each vertex of $\sigma_{L,i}^Q$ or $\sigma_{R,i}^Q$ add a corresponding point on line segments $\sigma_{L,i}^P$ or $\sigma_{R,i}^P$, respectively.

The line segments $\sigma_{L,i}^P$ and $\sigma_{R,i}^P$ together with γ_L^P and γ_R^P divide P into k simple polygons, $P_1, \ldots, P_k$. Similarly, the polygonal curves $\sigma_{L,i}^Q$ and $\sigma_{R,i}^Q$ together with γ_L^Q and γ_R^Q divide Q into k simple polygons, $Q_1, \ldots, Q_k$, corresponding to the subpolygons of P.

Both γ_L^P and γ_R^P have $O(n)$ vertices, so the k subpolygons P_i of P have a total of $O(n)$ vertices. Each polygonal curve $\sigma_{L,i}^Q$ and $\sigma_{R,i}^Q$ has $O(n)$ vertices, so each of the k subpolygons Q_i of Q has $O(n)$ vertices. Note that many of the vertices of Q_i will correspond to points on the boundary of P_i which are not vertices of P_i.

Let n_i be the number of vertices of subpolygon P_i. By Lemma 1, isomorphic triangulations of P_i and Q_i matching corresponding points on the boundaries of each can be constructed in $O(nn_i)$ time using $O(nn_i)$ triangles. Only the corresponding boundary points are used as boundary triangulation vertices, so the triangulations of P_i and Q_i can be patched together to give isomorphic triangulations of P and Q. Since the sum of the number of vertices n_i is $O(n)$, the isomorphic triangulations use $O(n^2)$ triangles.

Partitioning P by γ_L^P, γ_R^P, $\sigma_{L,i}^P$ and $\sigma_{R,i}^P$ takes $O(n)$ time. Constructing each $\sigma_{L,i}^Q$ and $\sigma_{L,i}^Q$ takes $O(n)$ time for a total of $O(n^2)$ time to subdivide Q. It takes $O(nn_i)$ time to construct isomorphic triangulations of the subpolygons P_i and Q_i and the sum of n_i is $O(n)$, so the total running time is $O(n^2)$. □

References

[1] ARONOV, B., SEIDEL, R., AND SOUVAINE, D. On compatible triangulations of simple polygons. *Comput. Geom. Theory Appl. 3*, 1 (1993), 27–35.

[2] GUPTA, H., AND WENGER, R. Constructing pairwise disjoint paths with few links. Technical Report OSU-CISRC-2/97-TR16, The Ohio State University, Columbus, Ohio, 1997.

[3] GUPTA, H., AND WENGER, R. Constructing piecewise linear homeomorphisms of simple polygons. *J. Algorithms 22* (1997), 142–157.

[4] HERTEL, S., AND MEHLHORN, K. Fast triangulation of simple polygons. In *Proc. 4th Internat. Conf. Found. Comput. Theory* (1983), vol. 158 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 207–218.

[5] KRANAKIS, E., AND URRUTIA, J. Isomorphic triangulations with small number of Steiner points. In *Proc. 7th Canad. Conf. Comput. Geom.* (1995), pp. 291–296.

[6] O'ROURKE, J. *Computational Geometry in C.* Cambridge University Press, 1994.

[7] PACH, J., SHAHROKHI, F., AND SZEGEDY, M. Applications of the crossing number. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.* (1994), pp. 198–202.

[8] SAALFELD, A. Joint triangulations and triangulation maps. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.* (1987), pp. 195–204.

[9] SOUVAINE, D., AND WENGER, R. Constructing piecewise linear homeomorphisms. Technical Report 94–52, DIMACS, New Brunswick, New Jersey, 1994.

On Folding Rulers in Regular Polygons

(extended abstract)

Naixun Pei* and Sue Whitesides†
McGill University‡

Abstract

An *l-ruler* is a sequence of n rigid rods lying in the plane and joined consecutively at their endpoints. The endpoints are joints about which the rods may freely turn, possibly crossing over one another. Each rod has length l. An l-ruler is said to be confined inside a polygon P if each link of the ruler must remain inside the closed, bounded region bounded by P at all times. An l-ruler confined inside P is said to be *always-foldable* if, for each possible initial configuration of the ruler, the ruler can be folded onto a single segment of length l.

A study of l-rulers confined to equilateral triangles was carried out by van Kreveld, Snoeyink and Whitesides, who showed that always-foldability is a property that alternates *four* times between holding and failing as l grows from 0 to its maximum possible value. They asked whether this phenomenon occurs for l-rulers confined inside other polygons.

The present paper extends their study to regular polygons. In particular, it answers their question in the affirmative: in regular $2k$-gons, the always-foldability of l-rulers alternates between holding and failing *three* times as l grows from 0 to its maximum possible value.

1 Introduction

A *chain* Γ is a sequence of n rigid rods (also called links) joined consecutively at their endpoints (also called joints), about which they may freely rotate. An l-ruler L is a chain whose links all have the same length l.

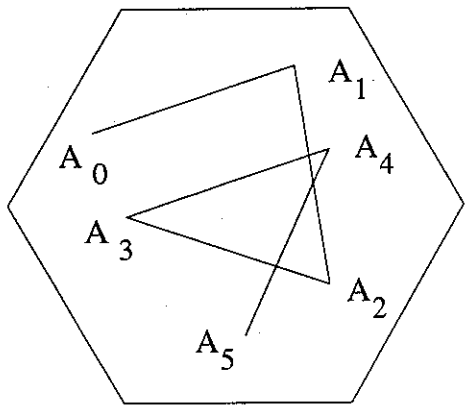


Figure 1: An l-ruler confined in a regular hexagon

A chain Γ is said to be *confined inside a polygon* P if its links must always lie inside the closed, bounded polygonal region determined by P. Two configurations of a chain Γ confined inside a closed polygonal region P are said to be *equivalent* if one can continuously move to the other while the links remain within P and while the lengths of the links maintain their initial values at all times.

Suppose an l-ruler L has just one equivalence class of configurations under the above notion of equivalence. Then in particular, every arbitrary configuration of L is equivalent to one in which all the links coincide. In this case, we say that L is *always-foldable*. Otherwise, when L has more than one equivalence class of configurations, we say that L is *not-always-foldable*.

This paper studies always-foldability for l-rulers confined inside a regular polygon P.

*pei@muff.cs.mcgill.ca
†sue@cs.mcgill.ca Supported by FCAR and NSERC.
‡School of Computer Science, 3480 University St. #318, Montreal, Quebec H3A 2A7 CANADA

Whether or not such an l-ruler is always-foldable depends in part on the relationship between l and w, the *width* of P. Here, recall that the width of any polygon, regular or not, is the minimum possible distance between two parallel lines of support for the polygon.

1.1 Summary of Main Results

Our main results are as follows. Let L be an l-ruler confined inside a regular $2k$-gon P. Then L is always-foldable for $l \leq w$; L is not-always-foldable for $w < l \leq m$, where m is the distance between a vertex of P and the midpoint of either of the two sides of P farthest from the vertex; finally, L is again always-foldable for $m < l \leq d$, where d is the diameter of P. The paper [5] proved that equilateral triangles exhibit more than one alternation of the always-foldability property and asked whether any other polygons exhibit this phenomenon. Hence, we answer this question in the affirmative, as we show regular $2k$-gons exhibit multiple alternation. Note that one would in general expect at least one transition, from always-foldable, for sufficiently small values of l, to not-always-foldable for larger values of l.

For regular $(2k+1)$-gons, we show that L is always-foldable for $l \leq b^C$, where b^C is the supremum of the radii of circles that, no matter where their centers are placed on P, cut P in exactly two places. For $w < l \leq d$, L is always-foldable. However, for $b^C < l \leq w$, we do not know for which, if any, values L is always-foldable. We leave this as an open problem.

1.2 Background

Reconfiguration properties of chains have been considered for example in [1], [2], [3], [4], [8], and [9]. In [7], the number of equivalence classes of unconfined closed chains (i.e., $A_n = A_0$) in arbitrary dimension is determined. Chains confined inside a circle and having an extremal joint anchored were studied in [1] and in [2]. Anchored and unanchored chains confined inside a square were studied in [3] and [4]. Unanchored chains whose links all have the same length (i.e., l-rulers) and that are confined inside an equilateral triangle were studied in [5]; anchored l-rulers were studied in [10]. Chains are themselves special cases of planar linkages, surveyed in [11].

1.3 Terminology

We say that a chain Γ is *bounded* by b, denoted by $\Gamma \prec b$, if no link has length greater than b.

A convex obtuse polygon is a convex polygon with all internal angles measuring $\pi/2$ or more. We denote by S a square with unit side length. We denote the distance between two points p, q by $|pq|$.

We regard a polygon P as a closed, polygonal curve bounding a two-dimensional, region of finite area. When we are referring to the closed curve and not to the region it bounds, we use the notation ∂P for emphasis.

For a chain Γ confined inside a polygon P, we say that Γ is in *Rim Normal Form* (denoted RNF), if all joints of Γ lie on ∂P.

In addition to the width w and diameter d of a polygon, we also make use of the lengths b^C (defined for arbitrary polygons) and m (defined for regular $2k$-gons); recall the definitions for m and b^C from section 1.1.

2 l-Rulers in a Square

In this section, we consider the case of a square S of unit side. For $2k$-gons with $k > 2$, covered in a later section, the proofs will be essentially the same.

2.1 Short Links

Here we prove that any l-ruler L with $l \leq 1$ is always-foldable. The key idea is to begin by moving L to Rim Normal Form (RNF). The following Fact is from [4].

Fact 2.1 *If $l \leq 1$, then L inside S can be moved to RNF*

Always-foldability for $l \leq 1$ is an immediate consequence, as described below.

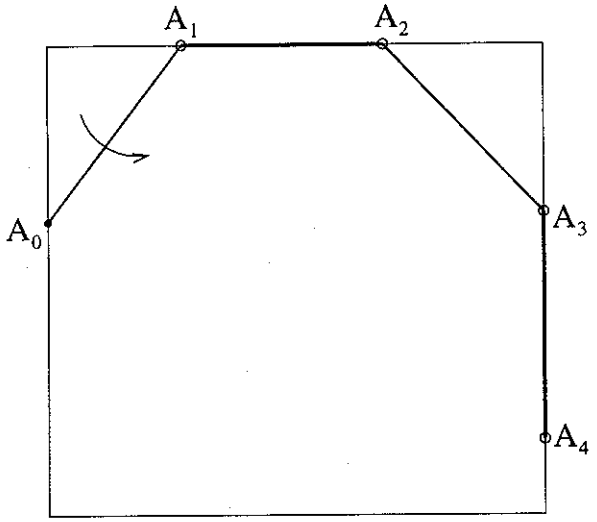


Figure 2: Always-foldable short-link rulers

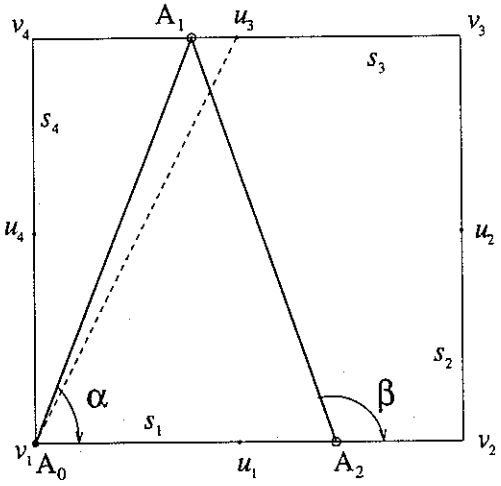


Figure 3: Not-always-foldable rulers

Theorem 2.1 *If $l \leq 1$, then L inside S is always-foldable.*

Proof: Move L to RNF in accordance with Fact 2.1. Then fold L inductively as follows.

Fixing $A_1, A_2, \ldots, A_n$, rotate $[A_0, A_1]$ about A_1 until A_0 and A_2 coincide, as Figure 2 shows. This is possible since $b^C = w = 1$ and $l < 1$. Continue this process until Γ folds. □

2.2 Midrange Links

This subsection shows that l-rulers inside S with $1 < l \leq \sqrt{5}/2$ are not-always-foldable. Note that $\sqrt{5}/2$ is the distance between a vertex of S and the midpoint of either of the two sides of S that are non-incident with the vertex (note the dashed line in Figure 3).

Theorem 2.2 *If $1 < l \leq \sqrt{5}/2$, then L inside S is not-always-foldable.*

Proof: Let s_1, s_2, s_3, s_4 be the sides of S, let v_1, v_2, v_3, v_4 be the vertices of S, and let u_1, u_2, u_3, u_4 be the midpoints of s_1, s_2, s_3, s_4, respectively. Suppose that α, β are angles between $[A_0, A_1]$, $[A_1, A_2]$ and s_1, respectively, as shown in Figure 3.

Initially we put L in the following configuration: A_0 lies at v_1, A_1 lies on s_3, A_2 lies at a point on s_1 but different from v_1. See Figure 3. We show that L cannot be folded from this configuration.

Since $1 < l \leq \sqrt{5}/2$, initially A_1 lies between v_4 and u_3. Therefore $\beta > \pi/2$ initially. Clearly $\alpha < \pi/2$ initially. If L is foldable, let γ be the angle between s_1 the line through some segment onto which L can be folded. Without loss of generality, assume $r \leq \pi/2$. Since initially $\beta > \pi/2$, there exists some intermediate configuration in which $\beta = \pi/2$, i.e., $[A_1, A_2]$ is perpendicular to s_1. This contradicts $l > 1$. □

2.3 Long Links

This subsection shows that any l-ruler inside S with $l > \sqrt{5}/2$ is always-foldable. We obtain this by proving that such rulers initially have to lie in "nearly-folded" configurations. In the extreme case of $l = \sqrt{2}$, the diameter of S, L has to exhibit an already folded configuration.

Before proceeding to the foldability result, we give some preliminaries. Again, let s_1, s_2, s_3, s_4 be the sides of S, let v_1, v_2, v_3, v_4 be the vertices of S, and let u_1, u_2, u_3, u_4 be the midpoints of s_1, s_2, s_3, s_4, respectively. Define C_1 as the area delimited by $v_1u_3, u_3v_3, v_3u_2, u_2v_1$, as shown in Figure 4. C_2, C_3 and C_4 are similarly defined. Then we have the following.

Lemma 2.1 *If $l > \sqrt{5}/2$, then L inside S falls completely inside exactly one of C_1, C_2, C_3, C_4.*

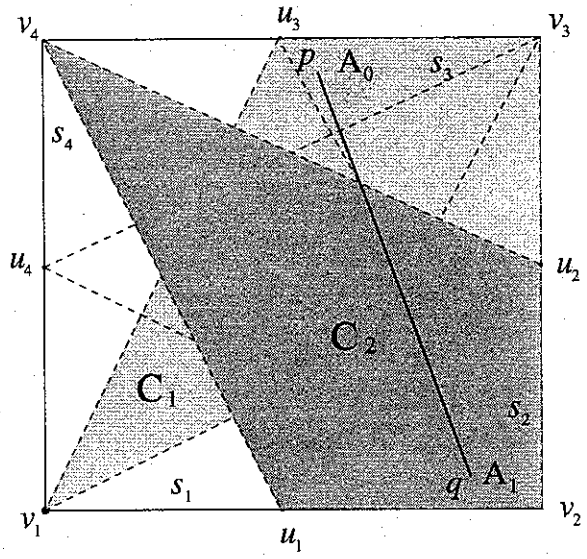


Figure 4: L lies inside one of C_1, C_2, C_3, C_4.

Proof: Since $C_1 \bigcup C_2 \bigcup C_3 \bigcup C_4 = S$, L lies inside $C_1 \bigcup C_2 \bigcup C_3 \bigcup C_4$. Next we show that L lies inside exactly one of C_1, C_2, C_3, C_4.

Note that for any i, j with $1 \leq i, j \leq 4, i \neq j$, $\forall p \in C_i, \forall q \in C_j$, $d(p, q)$ achieves its maximum at some vertex and the midpoint of some side. Therefore,

$$\max_{p \in C_i, q \in C_j} d(p, q) \leq \sqrt{5}/2.$$

Thus,

$$\max_{p \in C_i, q \in C_j - C_i} d(p, q) \leq \sqrt{5}/2. \qquad (*)$$

If there is a configuration in which L does not lie in any single one of C_1, C_2, C_3, C_4, then by convexity there exists i, j with $1 \leq i, j \leq 4, i \neq j$ and a link of L, say L_1, such that A_0 lies at some $p \in C_i$ and A_1 lies at some $q \in C_j - C_i$. See Figure 4. By $(*)$, $d(p, q) \leq \sqrt{5}/2$. This contradicts $l > \sqrt{5}/2$. □

Now we are ready to give the following foldability result.

Theorem 2.3 *If $l > \sqrt{5}/2$, then L inside S is always-foldable.*

Proof: By Lemma 2.1, L lies inside exactly one of C_1, C_2, C_3, C_4. Without loss of generality, assume that L lies inside C_1. We fold L inductively as follows. Fixing $A_1, A_2, \ldots, A_n$, rotate $[A_0, A_1]$ about A_1 until A_0 and A_2 coincide, as Figure 5 shows. We claim that A_0 will not hit the boundary ∂C_1 of C_1 during this reconfiguration.

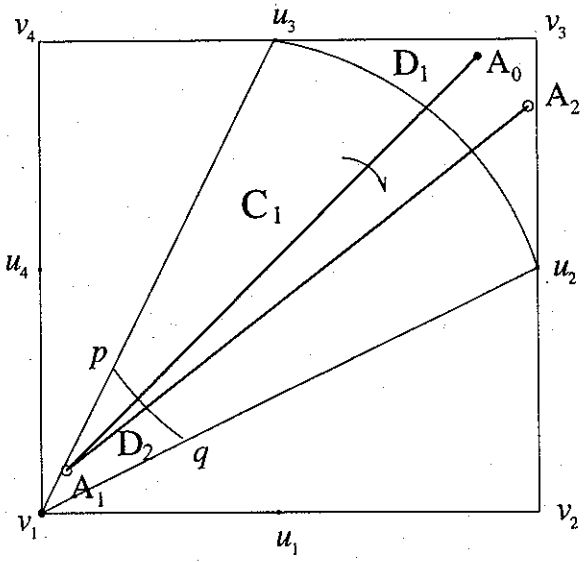


Figure 5: Folding long-link rulers

This can be seen as follows. Suppose that $\{p, q\} = C(v_3, \sqrt{5}/2) \bigcap \partial C_1$ and note that $\{u_3, u_2\} = C(v_1, \sqrt{5}/2) \bigcap \partial C_1$. We define cone D_1 as the area delimited by $v_3u_3, u_3\widehat{u_2}, u_2v_3$ and cone D_2 as the area delimited by $v_1p, \widehat{pq}, qv_2$, as shown in Figure 5. Since $l > \sqrt{5}/2$, each joint of L has to lie inside D_1 or D_2.

If A_0 lies in D_1, then A_1 lies in D_2 and A_2 lies in D_1. If A_0 lies in D_2, then A_1 lies in D_1 and A_2 lies in D_2. In both cases, A_0 will not hit ∂C_1 before A_0 and A_2 coincide. Hence the claim.

Continue this process until Γ folds. □

3 l-Rulers in a Regular $2k$-gon

This section generalizes the foldability result of an l-ruler within squares to arbitrary regular $2k$-gons.

3.1 Foldability within Regular $2k$-gons

The foldability result of an l-ruler within a square can be readily extended to any regular $2k$-gon, based on the following results from [9].

Fact 3.1 *Let Γ be an n-link chain confined within a convex obtuse polygon P. If $\Gamma \prec b^C$, then Γ can be moved to RNF.*

Fact 3.2 *Let P be a regular $2k$-gon. Then $b^C = w$.*

We thus have the following, which immediately implies the foldability of rulers with short links inside a regular $2k$-gon.

Fact 3.3 *Let P be a regular $2k$-gon. If $l \leq w$, then L can be moved to RNF.*

Theorem 3.1 *Let P be a regular $2k$-gon. If $l \leq w$, then L is foldable.*

The foldability of rulers with midrange and long links within a regular $2k$-gon is completely similar to that within a square. In the following, let P be a regular $2k$-gon, let v be a vertex of P, let u be the midpoint of an opposite side of v, and let $m = |uv|$.

Theorem 3.2 *Let P be a regular $2k$-gon. If $w < l \leq m$, then L is not-always-foldable. Refer to (a) of Figure 6.*

Theorem 3.3 *Let P be a regular $2k$-gon. If $l > m$, then L is always-foldable. Refer to (b) of Figure 6.*

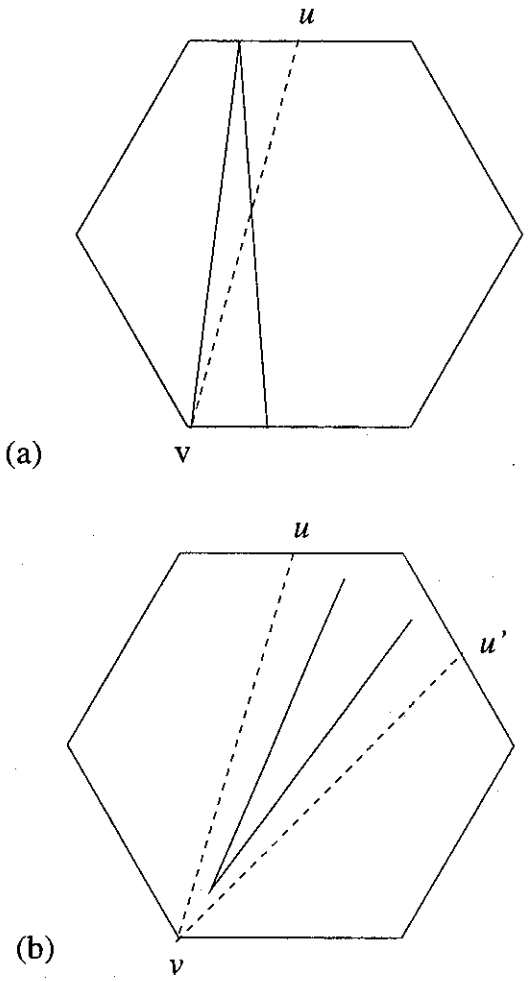


Figure 6: Foldability within regular $2k$-gons

4 Regular $(2k+1)$-gons

The reason that the above approach does not apply to regular $(2k+1)$-gons is due to gap between b^C and $w > b^C$ in regular $(2k+1)$-gons. For $b^C < l \leq w$, we do not know the foldability of L.

We observe that an l-ruler remains not-always-foldable when $l > w$, for reasons similar to the non-foldability of rulers with midrange links within a square. We use Figure 7 to suggest the ideas. This phenomenon shows that $2k$-gons and $(2k+1)$-gons as confining regions exhibit different foldability properties.

Below we give the known foldability results of rulers within regular $(2k+1)$-gons and pose the foldability of an l-ruler with $b^C < l \leq w$ inside regular $(2k+1)$-gons as an open problem.

Theorem 4.1 *Let P be a regular $(2k+1)$-gon. If $l \leq b^C$, then L is always-foldable.*

Theorem 4.2 *Let P be a regular $(2k+1)$-gon. If $l > w$, then L is not-always-foldable. Refer to Figure 7.*

5 Conclusion

As the segment length l of an l-ruler confined in a polygon P increases from 0 to its maximum value, one expects that for all sufficiently small l, the ruler is always-foldable and that for some critical value of l, the ruler become not-always-foldable. This paper has shown that in a regular $2k$-gon P, the always-foldability property alternates *three* times, from holding, to not

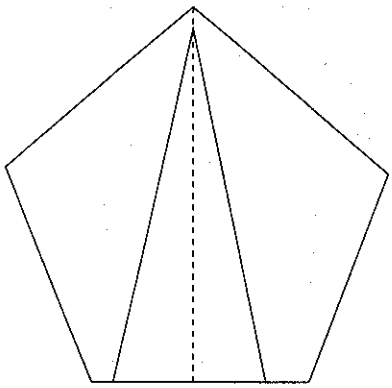


Figure 7: Foldability within regular $(2k+1)$-gons

holding, and finally back to holding. This answers in the affirmative the question of [5] as to whether interesting alternation phenomena occur for polygons other than equilateral triangles, where alternation occurs *four* times.

For regular $2k$-gons, we exhibited the critical values of l for which the transitions between always-foldability and not-always-foldability, or the reverse, occur.

For regular $(2k+1)$-gons, $k > 1$, we showed that all such transitions occur between b^C and w. We leave as an open problem the number of transitions that occur in this range, and their corresponding critical values.

References

[1] J. Hopcroft, D. Joseph and S. Whitesides. On the movement of robot arms in 2-dimensional bounded regions. *SIAM J. Comput.* **14 (2)**, pp. 315-333 (1985).

[2] V. Kantabutra and S. R. Kosaraju. New algorithms for multilink robot arms. *J. Comput. Sys. Sci.* **32**, pp. 136-153 (1986).

[3] V. Kantabutra. Motions of a short-linked robot arm in a square. *Discrete Comput. Geom.* **7**, pp. 69-76 (1992).

[4] V. Kantabutra. Reaching a point with an unanchored robot arm in a square. *Manuscript*, accepted for publication, *International J. of Computational Geometry and Applications.*

[5] M. van Kreveld, J. Snoeyink and S. Whitesides. Folding rulers inside triangles. *Discrete Comput. Geom.* **15**, pp. 265-285 (1996). A conference version appeared in Proc. of the 5th Canadian Conference on Computational Geometry, August 5-10 (1993), Waterloo, Canada, pp. 1-6.

[6] Jean-Claude Latombe. Robot Motion Planning. Kluwer Academic Publishers, Boston MA, USA (1991).

[7] W. Lenhart and S. Whitesides. Reconfiguring closed polygonal chains in Euclidean d-space. *Discrete Comput. Geom.* **13**, pp. 123-140 (1995).

[8] N. Pei and S. Whitesides. On the Reconfiguration of Chains *in* Proceedings of Cocoon '96, Hong Kong, June 17-19, 1996, Springer Verlag LNCS 1090, pp. 381-390.

[9] N. Pei. On the Reconfiguration and Reachability of Chains. Ph.D. Thesis, School of Computer Science, McGill University, November, 1996.

[10] I. Suzuki and M. Yamashita. Designing multi-link robot arms in a convex polygon. *International J. of Computational Geometry and Applications*, v. 6, no. 4, pp. 461-486 (Dec. 1996).

[11] S. H. Whitesides. Algorithmic issues in the geometry of planar linkage movement. *The Australian Computer Journal*, Special Issue on Algorithms, pp. 42-50 (May 1992).

On the number of internal and external visibility edges of polygons

Jorge Urrutia*
Department of Computer Science
University of Ottawa, Ottawa ON Canada

Abstract

In this paper we prove that for any simple polygon P with n vertices, the sum of the number of strictly internal edges and the number of strictly external visibility edges of P is at least $\lfloor \frac{3n-1}{2} \rfloor - 4$.

The *internal visibility graph* of a simple polygon P is the graph with vertex set equal to the vertex set of P, in which two vertices are adjacent if the line segment connecting them does not intersect the exterior of P. The *external visibility graph* of P is defined in a similar way, except that the line segments that generate its edges are not allowed to intersect the interior of P. A visibility edge is called *strictly internal* (resp. *strictly external*) if it is not an edge of P. In this paper we prove the following conjecture of Bagga [1]:

> For any simple polygon P with n vertices, the number of strictly internal visibility edges plus the number of strictly external visibility edges is at least $\lfloor \frac{3n-1}{2} \rfloor - 4$.

In Figure 1 we present a family of polygons that achieve this bound. They have exactly $n-3$ strictly internal visibility edges, and $\frac{n-3}{2}$ strictly external visibility edges.

*Supported by NSERC of Canada

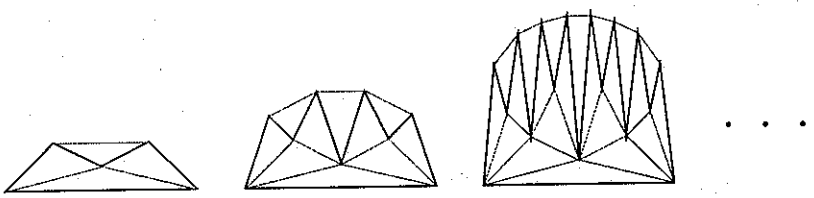


Figure 1: A sequence of polygons for which the number of strictly internal plus strictly external visibility edges is exactly $\lfloor \frac{3n-1}{2} \rfloor - 4$.

Let $int(P)$ and $ext(P)$ denote the number of strictly internal and external visibility edges of P. Some observations will be used to prove that for any polygon P with n vertices, $int(P) + ext(P) \geq \lfloor \frac{3n-1}{2} \rfloor - 4$. A vertex v of P will be called *internal* if it is in the *interior* of the convex hull $Conv(P)$ of P. An *external vertex* is a vertex of the convex hull of P.

The following result is easy to prove:

Lemma 1 *Let P be a simple polygon with n vertices, k of which are internal. Then $ext(P)$ is at least k.*

From this we have:

Lemma 2 *If P has k internal vertices, then $int(P) + ext(P) \geq (n-3) + k$.*

Proof: Observe that any triangulation of P has exactly $n-3$ strictly internal edges. By Lemma 1, $ext(P) \geq k$. ∎

We now prove:

Lemma 3 *If P has k internal vertices, then P can be decomposed into exactly $k+1$ convex polygons $P_1, \ldots, P_{k+1}$. Moreover this decomposition can be achieved in such a way that if n_i is the number of vertices of P_i, $i = 1, \ldots, k+1$, then $n_1 + \ldots + n_{k+1} = n + 3k$.*

Proof: One at a time, and for all the internal vertices v of P, repeat the following operation: starting at v, draw a line segment that bisects the internal angle of P at v and extend it until it hits the boundary of P, or a

previously drawn line segment. If this line segment hits a vertex of P, rotate it slightly so that it ends in the middle of an edge; see Figure 2. Observe that the endpoints of these segments, appear as vertices in exactly two of the resulting subpolygons of P, and therefore each of them contributes four units to $n_1 + \ldots + n_{k+1}$. Our result now follows. ∎

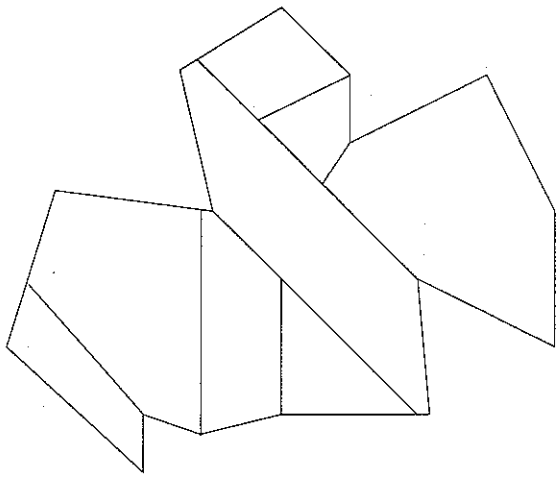


Figure 2: Partitionng a polygon into convex subpolygons.

In our previous lemma, each P_i has two types of vertices; those which are vertices of P, which we call *real* vertices, and vertices which are endpoints of the line segments used to partition P, and which are not vertices of P. Let P_i' be the convex polygon generated by the set of real vertices of P_i, and m_i be the number of vertices of P_i'. Notice that if $m_i \geq 4$, then any *strictly internal visibility edge of* P_i is intersected by at least $m_i - 3$ strictly internal visibility edges of P_i'. Thus we have:

Lemma 4 *If P_i' has m_i vertices, $m_i \geq 4$, then any* strictly internal visibility edge of P_i' *is intersected by at least $m_i - 3$ strictly internal visibility edges of P_i'.*

We now have:

Theorem 1 *For any simple polygon P with n vertices, $int(P) + ext(P) \geq \lceil \frac{3n-1}{2} \rceil - 4$.*

Proof: Suppose that P has k internal vertices. Partition it into $k+1$ convex polygons $P_i, \ldots, P_{k+1}$ as in Lemma 3. If $m_i \geq 4$, select a strictly internal visibility edge e_i of P_i', $i = 1, \ldots, k+1$. Obtain an internal triangulation T of P such that the set of edges e_i as defined before, belong to T. We now show that $int(P) \geq 2n - 2k - 6$. By Lemma 4, for each $m_i \geq 4$, edge e_i is intersected by at least $m_i - 3$ strictly internal edges of P_i'. Since e_i belongs to T, none of these edges belongs to T. Furthermore *these edges are strictly internal visibility edges of* P. It now follows that

$$int(P) \geq (n-3) + \sum_{m_i \geq 4} (m_i - 3) \geq (n-3) + \sum_{i=1,\ldots,k+1} (m_i - 3)$$

But

$$\sum_{i=1,\ldots,k+1} (m_i - 3) = n + k - 3(k+1) = n - 2k - 3$$

(each internal vertex of P appears in two P_i's and each vertex in the convex hull of P in one). Then we have

$$int(P) \geq 2n - 2k - 6$$

By Lemma 1, we know that $ext(P) \geq k$, and thus we have:

$$int(P) + ext(P) \geq 2n - k - 6$$

On the other hand, by Lemma 2 we have that

$$ext(p) + int(P) \geq (n-3) + k$$

Combining these equations we get that $int(P) + ext(P) \geq \lfloor \frac{3n-1}{2} \rfloor - 4$. ■

References

[1] J. O'Rourke, Combinatorics of visibility and illumination problems. *Technical report* 1996.

On a partition of point sets into convex polygons

MASATSUGU URABE *

Department of Mathematics, Tokai University,
3-20-1 Orido, Shimizu, Shizuoka, 424 Japan

April 5, 1997

ABSTRACT

In this paper three partitioning concepts are introduced. A partition of point sets in the plane is a convex partition if it separates the points into subsets such that each subset is a convex polygon. The results concern convex partitions of points sets with various conditions on the intersection properties.

1 BACKGROUND

Ester Klein [K](see also [ES1]) showed that from any five points in the plane, no three collinear, it is always possible to select four that comprise the vertices of a convex quadrilateral. She asked what the least integer $X(n)$ is such that from any $X(n)$ points in the plane, no three collinear, one can always select the vertices of a convex n-gon. That $X(n)$ exists was proved by Erdős and Szekeres [ES1] using Ramsey's theorem. A simple proof has been given by Johnson [J]. Klein result is X(4)=5. Kalbfleisch, Kalbfleisch and Stanton [KKS] proved that X(5)=9. A short proof was also given by Bonnice [B]. Erdős and Szekeres proved the following theorem.

Theorem (Erdoős-Szekeres theorem [ES2]).

$$2^{n-2}+1 \leq X(n) \leq \binom{2n-4}{n-2}$$

Any closing of the gap between these bounds would be significant. A conjecture is that $2^{n-2}+1$ is the correct value. So the first unsettled case is whether X(6)=17 or not. This conjecture can be equivalently stated as follows: every configuration of n points in the plane, no three collinear, contains the $\lfloor \log_2(n-1) \rfloor + 2$ vertices of a convex polygon.

Erdős [E] asked the following related question. What is the smallest integre $Y(n)$ such that from any $Y(n)$ points in the plane, no three collinear, one can choose the vertices of a convex n-gon, with none of the other points in its interior? We call a such convex polygon an *empty* convex polygon. Y(4)=5 is easy, and Harboth [H1] showed that Y(5)=10. Horton [H2] exhibited arbitrarily large sets containing no empty convex heptagons, so that $Y(n)$ does not exist for

* *e-mail address*: qzg00130@scc.u-tokai.ac.jp

$n \geq 7$. Thus the outstanding question is whether Y(6) exists, and if so what its value is. With the help of a computer Avis and Rappaport [AR] have found a set of 20 points, no three collinear, containing no empty hexagon. Moreover Fabella and O'Rourke [FO] found a similar such set of 22 points. Hence $Y(6) \geq 23$, if it exists.

Another result concerning a convex partition is Radon's theorem [R]: any $n \geq d+2$ points in $\mathbf{E}^d$ can be split into two subsets whose convex hulls have a common point. It is a fundamental result concern a convex partition of point sets with the intersection properties. In the next section we consider a related problem, namely, the partition of point sets in the plane into convex polygons [U1].

2 CONVEX PARTITION

Let P be a set of n points in the plane, no three collinear. A partition of P is called a *convex partition* if it separates P into k subsets $S_1, S_2, \ldots, S_k$; $\cup_{i=1}^{n} S_i = \mathrm{P}$, such that each $\mathrm{CH}(S_i)$ is a convex polygon the number of whose vertices is $|S_i|$, where $\mathrm{CH}(S_i)$ refers to the convex hull of the points set S_i. Then we say that the convex partition has *size* k. A partition $\Pi(\mathrm{P})$ is *disjoint* if each $\mathrm{CH}(S_i)$ is disjoint from the others; $\mathrm{CH}(S_i) \cap \mathrm{CH}(S_j) = \emptyset$ for any pair of i, j. Given a point set P, let $f(\mathrm{P})$ denote the minimum size of any disjoint partition of $\Pi(\mathrm{P})$. Define $F(n) = max\{f(\mathrm{P})\}$, over all sets P of n points in the plane. Our first problem is to determine $F(n)$.

Theorem 1.

$$\left\lceil \frac{n-1}{4} \right\rceil \leq F(n) \leq \left\lceil \frac{2n}{7} \right\rceil$$

Next, we consider a related problem. Given a point set P, let $\Pi'(\mathrm{P})$ be a convex partition of P such that each $\mathrm{CH}(S_i)$ is empty and let $g(\mathrm{P})$ be the minimum size of any $\Pi'(\mathrm{P})$. Define $G(n) = max\{g(\mathrm{P})\}$, over all sets of n points in the plane. Then we show the following:

Theorem 2.

$$\left\lceil \frac{n-1}{4} \right\rceil \leq G(n) \leq \left\lceil \frac{3n}{11} \right\rceil$$

We think that our upper bound for $G(n)$ is still fairly loose but we could not improve it to $n/4$. In fact there exists a configuration which satisfies $G(8) = 3$. Note that the inequality $f(\mathrm{P}) \geq g(\mathrm{P})$ holds for any set P of n points in the plane, that is, a set of disjoint convex polygons is also a set of empty convex polygons. Hence $F(n) \geq G(n)$ holds for every positive integer n. In addition, there exist several sets for which the inequality is strict.

The third problem we consider is the follwing. Given a set P of n points in the plane, no three collinear, estimate the minimum number of distinct convex polygons which cover P. Let $h(\mathrm{P})$ be the minimum size of any convex partition of P. Define $H(n) = max\{h(\mathrm{P})\}$, over all sets P of n points in the plane. Then we prove the following result.

Theorem 3.

$$\frac{n}{\log_2 n + 2} < H(n) < \frac{2n}{\log_2 n - \log_2 e}$$

3 3-DIMENSIONAL CASE

In the previous section, we studied the problem of partitioning point sets in the plane so that each equivalence class is a convex polygon disjoint from the others. We study here the case that 3-dimensional Euclidean space $\mathbf{E}^3$ of this problem [U2].

Let P be a set of n points in $\mathbf{E}^3$ which is in general position, no four points on a plane. A *m-polytope* is a convex polytope in $\mathbf{E}^3$ the number of whose vertices is m. A partition of P is called a *convex partition* if P is partitioned by k subsets $S_1, S_2, \ldots, S_k$ such that each $\mathrm{CH}(S_i)$ is a convex $|S_i|$-polytope. Let $\Pi^3(\mathrm{P})$ be a convex partition of P such that each $\mathrm{CH}(S_i)$ is disjoint from the others. Given a point set P, let $f^3(\mathrm{P})$ denote the minimum number of disjoint convex polytopes over all convex partitions $\Pi^3(\mathrm{P})$. Define $F(n) = max\{f^3(n)\}$, over all sets P of n points in $\mathbf{E}^3$. Moreover, let $g^3(\mathrm{P})$ be the minimum number of sets in any partition into empty convex polytopes of P. Define $G^3(n) = max\{g^3(\mathrm{P})\}$, over all sets P of n points. We now prove the following result.

Theorem 4.

$$\left\lceil \frac{n}{2(\log_2 n + 1)} \right\rceil \leq G^3(n) \leq F^3(n) \leq \left\lceil \frac{2n}{9} \right\rceil$$

4 REMARKS

The upper bound for $\mathrm{F}(n)$ can not be improved to $3n/11$. In fact, there exists a configuration which satisfies $F(11) = 4$. To prove the lower bound for $\mathrm{F}(n)$, we constructed the $2m$ points set $Q_m \cup Q'_m$ such that any polygon determined by three points of Q_m contains at least one point of Q'_m. Then the configuration requires $\lceil \frac{n-1}{4} \rceil$ disjoint (empty) convex polygons in any disjoint partition. For the 3-dimensional case, we conjecture that such a configuration exists; there exists a $2m$ points set $Q_m \cup Q'_m$ such that any 4-polytope in Q_m contains at least one point of Q'_m. Thus we conjecture that $F^3(n) = n/6$.

These problems can be generalized, in an obvious way, to higher dimensions. First, generalizing the definition of $F^3(n)$, let $F^d(n)$ denotes the maximum of the minimum number of disjoint convex polytopes in $\mathbf{E}^d$. Then we conjecture that for any set of n points in general position in $\mathbf{E}^d$, $d \geq 3$, $F^d(n) = n/2d$.

Another interesting problem is to study a similar question for empty convex sets. Given a point set P in $\mathbf{E}^d$, let $g^d(\mathrm{P})$ be the minimum number of sets in a partition into empty convex polytopes of P. Define $G^d(n) = max\{g^d(\mathrm{P})\}$, over all sets P of n points. What is the exact value for $G^d(n)$? In general, the inequality $F^d(n) \geq G^d(n)$ holds for every positive integer n. We know that $F^2(n) \geq G^2(n)$ for $n = 8, 11, 13$. It would be nice to generalize to other values of n, d.

We are also interested in the following application. Estimate the minimum number of distinct convex polytopes which cover a given point set in $\mathbf{E}^d$. Let $h^d(\mathrm{P})$ be the minimum number of subsets in any convex partition of a given point set P in $\mathbf{E}^d$. Define $H^d(n) = max\{h^d(\mathrm{P})\}$ over all sets P of n points in $\mathbf{E}^d$. An open problem is to find the exact value for $H^d(n)$.

ACKNOWLEDGEMENT

The autor would like to thank Professor David Avis for his valuable suggestions.

References

[AR] D.Avis and D.Rappaport, Computing the largest empty convex subset of a set of points, *Proc. First ACM Symp. on Computational Geometry*, Baltimore (1985) 161-167.

[B] W.E.Bonnice, On convex polygons determined by a finite planar set, *Ame. Math. Monthly*, 81, (1974) 749-752.

[E] P.Erdős, Some combinatorial problems in geometry, *Geometry and Differential Geometry (Proc. Conf. Univ. Haifa 1979)*, Lecture notes in Math. 792, Springer-Verlag, (1980) 46-53.

[ES1] P.Erdős and G.Szekeres, A combinatorial problem in geometry, *Compositio Math.*, 2, (1935) 463-470.

[ES2] P.Erdős and G.Szekeres, On some extremum problems in elementary geometry, *Ann. Univ. Sci. Budapest* 3/4, (1960/61) 53-62.

[FO] G.Fabella and J.O'Rourke, Twenty-two points with no empty hexagon, Manuscript, (1986).

[H1] H.Harborth, Konvexe Fünfecke in ebenen Punktmenger, *Elem. Math.*, 33, (1978) 116-118.

[H2] J.D.Horton, Sets with no empty convex 7-gons, *Canad. Math. Bull.*, 26, (1983) 482-484.

[J] S.Johnson, A new proof of the Erdős-Szekeres convex k-gon result, *J. Comb. Th.* (*A*), 42, (1986) 318-319.

[KKS] J.D.Kalbfleisch, J.G.Kalbfleisch and R.G.Stanton, A combinatorial problem on convex n-gons, *Proc. Louisiana Conf. on Combinatorics, Graph Theory and Computing*, Baton Rouge, (1970) 180-188.

[K] E.Klein, see Moser Wm., *Research problems in Geometry*, McGill Univ. ♯ 29, (1981).

[R] J.Radon, Mengen Konvexer Körper, die einen gemeinsamen Punkt enthalten, *Math. Ann.*, 83, (1921) 113-115.

[U1] M.Urabe, On a partition into convex polygons, *Discrete Applied Math.*, 64, (1996) 179-191.

[U2] M.Urabe, Partitioning point sets in space into disjoint convex polytopes, *to appear in Comput. Geom. Theory Appl.*.

Domino Tilings and Two-by-Two Squares
(Extended abstract)

Jurek Czyzowicz[§†] (czyzowicz@uqah.uquebec.ca)
Evangelos Kranakis[*†] (kranakis@scs.carleton.ca)
Jorge Urrutia[‡†] (jorge@csi.uottawa.ca)

Abstract

We consider the graph of domino tilings of simple polygons: two tilings are adjacent if one can be obtained from the other via a flip operation of the form ⊟ → ◫ or ◫ → ⊟. The graph of domino tilings of simple polygons is connected and we study the complexity of determining adjacency in this graph. We consider domino tilings of $n \times n$ squares. For $k = 1, 2$ we characterize the domino tilings of $n \times n$ squares which have exactly k nonoverlapping 2×2 squares. We also consider combinatorial enumeration problems related to such tilings.

1 Introduction

Polyominoes are shapes formed of equal sized squares (e.g. unit squares). They were introduced by S. W. Golomb [2, 1]. Dominoes are the simplest nontrivial polyominoes and consist of two unit squares adjacent along an edge.

We consider domino tilings of rectangles and squares (also known as checkerboards). Every domino tiling of a nontrivial rectangle has two dominoes forming a 2×2 square of the form ⊟ or ◫. We are interested in the problem of characterizing domino tilings of $n \times n$ squares with exactly k nonoverlapping 2×2 squares. We give such characterizations when $k = 1, 2$: if a domino tiling of an $n \times n$ square has exactly k nonoverapping squares then these squares must lie with a bounded subtiling located at the center of the square. In particular, there is a unique (up to rotation) domino tiling of an $n \times n$ square which has exactly one (respectively, two nonoverlapping) 2×2 square(s).

The graph of domino tilings of simple polygons is connected and we study the complexity of determining adjacency in this graph. In the next section we consider the distance function among tilings and study the complexity of computing the distance among tilings. For the case of $n \times 2, n \times 3$ it is even possible to derive recursive expressions. In the sequel we consiser combinatorial formulas on the number of tilings with a given number of nonoverlapping 2×2 squares.

A related study on polyomino tilings is in [3]. Combinatorial studies on domino tilings can be found in the book [4]. Extensive studies and literature on tilings can be found in [5]. Throughout this paper we assume that the dominoes are located at integer lattice points.

2 2×2 Squares in Rectangles

In this section we concentrate on proving that every domino tiling of a rectangle with both sides > 1 must have a (subtiling consisting of a) 2×2 square. All definitions below refer to a domino tiling of a given rectangle.

DEFINITION 1 *An L-domino (respectively, R-domino) configuration is a vertical domino with a horizontal domino attached either to its right (respectively, left) or below. The base of an L-domino (respectively, R-domino) configuration is the y-coordinate at the base of the horizontal domino attached to it.*

Thus among the four polyominoes ⊾ ⊿ ⌐ ⌋ the two configurations to the left are L-dominoes while the two configurations to the right are R-dominoes.

DEFINITION 2 *An (L, R)-domino pair is a pair consisting of an L-domino configuration and an R-*

§Département d'Informatique, Université du Québec à Hull, Hull, Québec J8X 3X7, Canada.
*Carleton University, School of Computer Science, Ottawa, ON, K1S 5B6, Canada.
‡University of Ottawa, Department of Computer Science, Ottawa, ON, K1N 9B4, Canada.
†Research supported in part by NSERC (National Science and Engineering Research Council of Canada) grant.

domino configuration such that both configurations have identical bases. Moreover the configurations are nonoverlapping and the L-domino configuration lies to the left of the R-domino configuration.

This gives rise to four (L,R)-domino pairs ⊢ ⊣, ⊢ ⫞, ⫟ ⊣, ⫟ ⫞. By rotating the configurations $90, 180, 270$ degrees it is clear that we may well define four types of (L,R)-domino pairs. Results below are valid for all four types but for convenience we mention the proofs only for the (L,R)-domino pair corresponding to the horizontal.

DEFINITION **3** *For a given (L,R)-domino pair whose vertical dominoes have x-coordinates $0, a$ respectively, the pyramid polygon is the polygonal line delimited by the horizontal straight line segment determined by the points $(0,0),(a,0)$, the points $(\lfloor a/2\rfloor-1, \lfloor a/2\rfloor+2), (\lfloor a/2\rfloor+1, \lfloor a/2\rfloor+2)$ and the line segments determined by tracing the points (see Figure 1)*

$$(0,0),(0,3),(1,3),(1,4),\ldots,(\lfloor a/2\rfloor-1,\lfloor a/2\rfloor+2)$$

and

$$(a,0),(a,3),(a-1,3),(a-1,4),\ldots,(\lfloor a/2\rfloor+1,\lfloor a/2\rfloor+2).$$

The definition above can be easily adapted to any (L,R)-domino pair as above. The following Lemma will be very useful for all our subsequent considerations.

LEMMA **4 (The Pyramid Lemma)** *Suppose that a domino tiling of a rectangle and an (L,R)-domino pair is given. Then there is a 2×2 square within the pyramid polygon formed by an (L,R)-domino pair.*

PROOF. There are two L-domino and two R-domino configurations. This give rise to four possible (L,R) -domino pairs. It is easy to see that it is enough to consider only one such domino pair, namely ⊢ ⊣. (We leave it to the reader to find the elementary transformations that reduce to this case.)

Starting with the dominoes L, R we search for a 2×2 square in the domino tiling by alternating between left and right side of the pyramid polygon determined by the given (L,R)-domino pair. Look at the "vacant" square position immediately to the right of the vertical domino in the L-domino configuration. If this is occupied by a horizontal domino then the 2×2 square has been found and the proof is complete. Otherwise this square must be occupied by a vertical domino. If the position which is below and to the right of this domino is occupied by a vertical domino then again the 2×2 square has been found and the proof is complete. Otherwise this square must be occupied by a horizontal domino. Next imitate this argument with domino R. Look at the "vacant" square position immediately to the left of the vertical domino of the R-configuration. If this is occupied by a horizontal domino then the 2×2 square has been found and the proof is complete. Otherwise this square must be occupied by a vertical domino. If the position which is below and to the left of this domino is occupied by a vertical domino then again the 2×2 square has been found and the proof is complete. Otherwise this square must be occupied by a horizontal domino. This idea is illustrated in the transformation ⊢ ⊣ → ⫞ ⫟, which also illustrates the step-by-step formation of the pyramid.

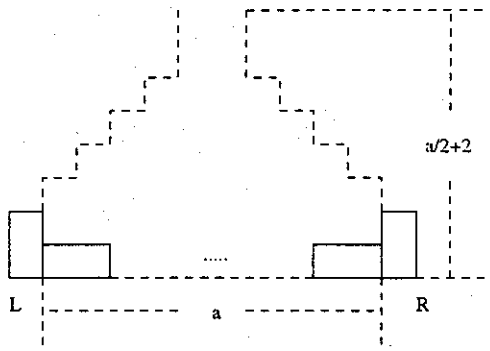


Figure 1: The pyramid of a tiling determined by an (L,R)-domino pair. If a is the distance of the vertical dominoes of the (L,R)-domino pair then a 2×2 square is guaranteed to exist within the pyramid depicted. The height of the pyramid is at most $\lfloor a/2\rfloor+2$.

This gives rise to a new (L,R)-domino pair. Now we iterate this procedure by alternating left-to-right forming a puramide like structure within which a 2×2 square is guaranteed to exist. The pyramid structure formed is depicted in Figure 1. The proof of the Lemma is now complete. ■

There are several possible extensions of the proof of the pyramid Lemma. For eaxample, starting with any of the configurations ⊢ ⫞ ⊣ ⫟ alone we can iterate the above idea of "filling spaces"in order to guarantee the existence of a 2×2 square in the space delimited by the domino configuration and the perimeter of the rectangle. A variant of this argument will also be used in the proof of Theorems 8, 9. Details of this are left to the reader. An interesting application is the following Theorem.

THEOREM **5** *Domino tilings of rectangles with both sides of length >1 must always have a 2×2 square.*

PROOF. Let the given rectangle be of dimension $m \times n$. The proof is by induction on the area mn of the rectangle. The result is easy to prove by inspection if either m or n is equal to 2. Hence we may assume without loss of generality that both $m, n > 2$. A simple area argument shows that either m or n is even. Without loss of generality we may assume the horizontal dimension m is even. If there is no horizontal domino lying along the horizontal touching the perimeter then we can peel off a layer from the rectangle and reduce to a rectangle of dimension $m \times (n-1)$. Since the resulting rectangle has area $m(n-1) < mn$, the theorem follows by induction. Hence without loss of generality we may assume there is a side with at least two vertical dominoes. Choose two such adjacent dominoes. Then the dominoes between them and adjacent to the perimeter must be all horizontal. Hence Lemma 4 implies the desired result. ■

As a matter of fact we can give a more precise count of the number of 2×2 squares.

THEOREM 6 *Domino tilings of rectangles of dimension $m \times n$, where $1 < m \leq n$, must always have at least $\lfloor n/(m+1) \rfloor$ nonoverlapping 2×2 squares.*

PROOF. We use the pyramid lemma. If there is a vertical domino at level $y = 0$ then we search for a 2×2 square starting either with an L- or with an R-domino configuration. Since $m \leq n$ such a 2×2 square must exist within $y \leq m$. If there is no vertical domino at $y = 0$ then we look for a vertical domino at $y = 1$. If such a vertical domino exists then again arguing as before we find a 2×2 square within $y \leq m + 1$. However, if there is no vertical domino either at level 0 or at level 1 then there is a 2×2 square within $y \leq 1$. In either case we can always find a 2×2 square within $y \leq m + 1$. Now we continue this process searching for a 2×2 square within $m + 2 \leq y \leq 2(m + 1)$, and so on. Iterating we obtain the desired result. ■

As a corollary we also obtain that if the rectangle has a domino tiling with at least k nonoverlapping 2×2 squares then $n \leq (k + 1)(m + 1)$.

The lower bound $\lfloor n/(m+1) \rfloor$ is attained as shown by the following example. Tile an $(3k + 1) \times 2$ rectangle iterating k copies of a vertical domino followed by two horizontal adjacent dominoes (i.e. of the form ⊟) and ending in a vertical domino. The resulting tiling has exactly k nonoverlapping 2×2 squares.

3 Tilings of Squares

In the sequel we consider domino tilings of $n \times n$ squares which have a given fixed number of 2×2 squares. Our main theorem gives precise characterizations for such tilings when the number of nonoverlapping 2×2 squares is either 1 or 2. A feature of our result is the existence of a "bounded" (i.e. independent of n) subtiling of the original tiling with the the specified number of 2×2 squares. In particular we prove the following theorem.

THEOREM 7 *Let T be a domino tiling of an $n \times n$ square which has exactly k nonoverlapping 2×2 squares. For each $k \leq 2$ there is a subtiling of T located at the center of the $n \times n$ square and forming a $2k \times 2k$ square and which has exactly k nonoverlapping 2×2 squares.*

As immediate corollaries we obtain that for each n even there is a unique (up to rotation) domino tiling of an $n \times n$ which has a unique 2×2 square (respectively, exactly two nonoverlapping 2×2 squares). The two unique domino tilings for the cases $k = 1, 2$ are depicted in Figure 2. In the sequel we give the proof of the theorem by considering each of the two cases, separately.

PROOF (OUTLINE). of Theorem 7.

Case 1: Tilings with exactly one 2×2 square

Let us suppose we are given a tiling T which has a unique 2×2 square. Following the details of the proof it will be shown that T must have a certain canonical representation which is unique.

The case $n = 2$ is trivial. So without loss of generality we may assume that n is an even integer > 2. The tiling covers the entire square. Moreover it is easy to see that there must exist a side which has dominoes adjacent and perpendicular to it. Also n is even, hence the number of such dominoes must also be even. Consider a side of the square with exactly two dominoes perpendicular to it. (See Figure 3, where we are assuming, without loss of generality, that the side under consideration is horizontal.) First of all we prove that these two dominoes cannot be adjacent. If they were then we would have two possibilities as depicted in parts (A) and (B) of Figure 3. All the dominoes, but two, adjacent to this side must be horizontal. Consider the unit square above the rightmost square position. This is covered either by a horizontal domino (resulting in part (A) of the Figure) or by a vertical domino (resulting in part (B) of the Figure). In part (A) this gives rise to two 2×2 squares, which is contradiction. In part (B) we can use the pyramid Lemma to produce a second 2×2 square which gives a contradiction. It follows that the two dominoes cannot be adjacent.

We claim that both vertical dominoes must be adjacent to corners of the square. If not then at least one of the squares is not adjacent. This gives rise to

either of the two configurations depicted in parts (C), (D) of Figure 3. However in both cases it is easy to see that by using the pyramid lemma we can show that the tiling must have at least two 2×2 squares, which is a contradition. It is also straightforward to see that there is no side of the square with more than two vertical dominoes, since in this case the tiling must have more than one 2×2 square.

It follows easily that two opposite sides of the $n \times n$ square have two dominoes each adjacent to their corners while the other two sides have only dominoes which are not perpendicular to them. Consequently, we can peel off a layer of the $n \times n$ square to produce a new $(n-2) \times (n-2)$ square which is tiled by the remaining dominoes of the tiling. Iterating this idea and by peeling off dominoes one layer at a time we reduce to a unique 2×2 square. This completes the proof of the theorem in this case.

Case 2: Tilings with exactly two nonoverlapping 2×2 squares

Let us suppose we are given a tiling T which has exactly two nonoverlapping 2×2 squares. Following the details of the proof it will be shown that T must have a certain canonical representation which is unique.

The proof considers several possible configurations. First of all consider the case where one side (say the horizontal) of the square has four dominoes perpendicular to it. These dominoes may be adjacent in which case they form groups by the number of adjacent dominoes in the group. If they form more than two separate groups then the theorem is immediate from the pyramid lemma. So consider the case they form two groups or less. One case is when one group has all four dominoes (depicted as parts (A) and (B) in Figure 4). Another case is when one group has three dominoes and the other one (depicted as parts (C), (D), (E), (F) in Figure 4). The last case is when there are two groups each consisting of two dominoes (depicted as parts (G), (H), (J) in Figure 4). In all these cases it is straightforward to see using the pyramid lemma that there exist at least three nonoverlapping 2×2 squares in the given tiling.

In particular it follows that no side of the square can have more than two dominoes perpendicular and adjacent to it. This reduces to either of the four configurations depicted in parts (A), (B), (C),(D) of Figure 3. A careful analysis of this shows that the tiling either will have more than two nonoverlapping 2×2 squares or we can peel off a layer of dominoes from the original square; the resulting square must be a 4×4 square whose domino tiling has exactly two nonoverlapping 2×2 squares. (See Figure 2.) This completes the proof of the theorem in this case. ■

4 Tilings of Simple Polygons

We consider the set $\mathcal{T}$ of domino tilings of a simple polygon. We define a graph on the set of tilings as follows. The set of vertices is $\mathcal{T}$ and two tilings are adjacent if one can be obtained from the other via a flip of the form $\square\square \rightarrow \square$ or $\square \rightarrow \square\square$. The resulting graph is bipartite. (To see this note that there are two kinds of tilings depending on the parity of the number of squares of the form $\square$.) We prove now that the graph of tilings of a given polygon P is connected, i.e. any given tiling of P may be obtained from any other tiling of P via a finite sequence of flips.

Suppose that P has been embedded in a chessboard. Obviously, if P is tilable, exactly half of the cells of P must have been embedded on the black cells of the chessboard. We will suppose in the proof, that each cell of P is either black or white, according to such an embedding.

THEOREM 8 *The graph of tilings is connected.*

PROOF. Let P denote a minimal polygon, and let T and T' denote two tilings of P such that T' cannot be obtained via a sequence of flips starting at T. Construct a directed graph G having a vertex for each cell of P. The edges of G are of two colors: red and blue. For each domino piece used in T, which covers two cells of P - a black cell c_1 and a white cell c_2, construct a red edge of G, going from c_1 to c_2. Similarly, for each domino piece used in T', which covers a black cell c_1 and a white cell c_2, construct a blue edge of G, going from c_2 to c_1 (see Figure 5). Observe that G is a planar, bipartite graph with vertices of degree 2. G must then be a union of disjoint, planar cycles. Each cycle is of an even size, formed using edges of alternate colors.

Observe. that all the boundary cells must belong to the same cycle. Indeed, if two boundary cells c_i and c_j had belonged to two disjoint cycles, the graph G could have been partitionned into at least two planar components, each one containing a subset of cycles of G. Polygon P could then be divided into at least two sub-polygons, such that the two tilings of one of these sub-polygons are also not connected by a sequence of flips. This would contradict assumed minimality of P.

Note that all the boundary cells must be traversed by the cycle of G containing them in the same order, i.e. either clockwise or counter-clockwise. Indeed, if this is not the case, there exist two boundary tiles (i.e. c_1c_2 and c_3c_4 in Figure 5), one contributing a clockwise edge to the boundary cycle, and another one contributing a counter-clockwise edge. It is then

not possible to form a planar cycle containing them (indeed, it is not possible to construct two disjoint paths within P, one from c_2 to c_3 and another one from c_4 to c_1).

From polygon P take the bottom row of cells. Let c_1 denote the leftmost cell of this row. Observe that the cell c_2, right to c_1, does also belong to P. Otherwise, the tile covering c_1 is the same in T and T' and we can eliminate from P both cells covered by it thus contradicting minimality of P.

There are two cells adjacent to c_1 in P. As T and T' must be different on boundary cells, suppose, by symmetry, that c_1 is covered by a vertical cell in T, and by a horizontal cell in T' (see Figure 6 (b)). Consider now a tile t_2 of T covering cell c_2. If t_2 is vertical, we can flip t_1 and t_2, placing both tiles in horizontal position, one of them beeing the same as tile t'_1 in T'. We can eliminate from consideration two cells covered by tile t'_1, contradicting minimality of P. Thus c_2 must be covered in T by a horizontal tile t_2. Consider now cell c_3 in the corner formed by t_1 and t_2 (see Figure 6 (c)). The tile t_3 covering c_3 cannot be horizontal, otherwise t_2 and t_3 could be flipped to vertical position, then, in turn, t_1 could be flipped, again obtaining the situation contradicting minimality of P. By induction we are forced to create an alternating sequence of vertical and horizontal tiles in T until we reach the boundary of P (see Figure 6 (d)). The corner cell c_n belongs then to the exterior of P. Observe, that tile t_1 contributes a counter-clockwise edge to the boundary cycle, while the last tile of the alternating sequence (t_{n-1} in Figure 6 (d)d) contributes a clockwise edge, contradicting an earlier observation. ■

Let $w = \text{width}(P)$, $h = \text{height}(P)$. Without loss generality assume $w \geq h$. From the proof of the latest theorem follows.

THEOREM 9 *In $O(wh^2)$ time we can find a sequence of flips which will convert one given tiling T to another tiling T' of a polygon P.*

PROOF (OUTLINE). We subdivide P into subpolygons corresponding to planar cycles of the graph G as in the proof of the previous theorem. At each step of the algorithm we take into consideration a boundary cell c_1 (say the leftmost cell of the bottom row) of one of such subpolygons. We traverse an alternating sequence of tiles until we detect a flipable pair. Then we perform a sequence of flips until we obtain the same boundary tile for both tilings T and T'. We then remove the two cells covered by such tile from P and we repeat the process for the remainder of the polygon. The total complexity follows from the fact that a removal of a pair of cells takes $O(h)$ time, and that there is at most $O(wh)$ such cells in P. ■

5 Tilings of Rectangles

In this section we study computational and combinatorial methods for the distance function among rectangular tilings.

5.1 Distance among tilings

Consider the graph of domino tilings. How do you compute the distance of any two domino tilings in this graph? Let $d(T, T')$ denote the distance between T, T'. There are two kinds of domino tilings: those whose two leftmost squares are occupied by a vertical domino ▯ and those occupied by ▭. It can now be argued that the following theorem holds.

THEOREM 10 *The distance function between any two domino tilings of an $n \times 2$ rectangle can be computed from the following recursive identities.*

1. $d(▯T, ▯T') = d(T, T')$.
2. $d(▭T, ▭T') = d(T, T')$.
3. $d(▯T, ▭T') = 1 + d(T, ▯T')$. ■

Although we omit details of the proof we urge the reader to delve into its elegant subtleties. This theorem easily gives a linear algorithm for computing the distance function. The same analysis will work for $n \times 3$ checkerboards except that there are more cases to consider.

5.2 Number of tilings

In Graham et al [4] $\sum_{n\geq 0} f(n)x^n = \frac{1}{1-x-x^2}$. is shown to be the generating function equation for $f(n) =$ the number of domino tilings of an $n \times 2$ rectangle. Moreover, the number of domino tilings of an $n \times 2$ rectangle with exactly s squares of the form ▭ is shown to be equal to $\binom{n-s}{s}$. In the sequel we compute the generating functions for the number of nonoverlapping 2×2 squares and squares of the form ⊡. We can prove the following theorems.

THEOREM 11 *Let $f(n, s)$ be the number of domino tilings of an $n \times 2$ rectangle with exactly s nonoverlapping 2×2 squares of the form ▭ or ⊡. Then*

$$\sum_{n,s\geq 0} f(n,s)x^n y^s = \frac{y(2x^2 + 3x^3 + x^4)}{1 - 2x^2y - x^3y}.$$

Moreover for $n \geq 8$ or $s \geq 3$, $f(n, s) =$

$$2^{3s-n}\left(\binom{s-1}{3s-n-1} + 3\binom{s-1}{3s-n} + 2\binom{s-1}{3s-n+1}\right).$$

THEOREM **12** *Let $f(n,s)$ be the number of domino tilings of an $n \times 2$ rectangle with exactly s nonoverlapping squares of the form* ⊞. *Then*

$$\sum_{n,s\geq 0} f(n,s)x^n y^s = \frac{x^2+x^2y}{1-x^2-x^3-x^2y}.$$

Moreover

$$f(n,s) = \sum_{\substack{n=2(r+1)+m \\ m+s\leq r+1}} \binom{r}{r-m}\binom{r-m+1}{s}.$$

PROOF (OUTLINE). of Theorems 11, 12. Let t_n^k denote the number of domino tilings of an $m \times 2$ rectangle having exactly k nonoverlapping 2×2 squares. We note that the two leftmost squares of a domino tiling may be occupied with either of the following three configurations ⊟, ⊞, ⊞. The remaining tiling has exactly $k-1$ nonoverlapping squares. Therefore this gives rise to the following recursive formula $t_n^k = 2t_{n-2}^{k-1} + t_{n-3}^{k-1}$. Using the obvious initial conditions $t_1^1 = 0, t_2^1 = 2, t_3^1 = 3, t_4^1 = 1, t_5^1 = 0$ we can verify easily the generating function in the first theorem. The proof of the second theorem is similar. Let f_n^k denote the number of domino tilings of an $m \times 2$ rectangle having exactly k nonoverlapping squares of the form ⊞. The resulting recursive formula is $f_n^k = f_{n-2}^k + f_{n-3}^k + f_{n-2}^{k-1}$. In both theorems it is now easy to derive precise formulas for the coefficients of the infinite series by expanding the generating function into infinite series [9]. We leave the details to the reader. ■

6 Conclusion and Open Problems

It is also possible to study the case of exactly three nonoverlapping squares but the characterizations are not so elegant. The situation is different for the case of tilings of an $n \times n$ square with exactly four nonoverlapping 2×2 squares, becuase these need not be located within a bounded subtiling of the original tiling. An interesting question left open for further investigation concerns the characterization of domino tilings which have a given "polyomino" pattern.

A sophisticated study (using homotopy theory) on the distance function can be found in [7], however no precise algorithmic analysis of the complexity of this problem has ever been carried out for arbitrary checkerboards.

Another interesting question concerns finding an algorithm to determine whether a given graph G is the bipartite graph of the set of domino tilings of an $m \times 2$ (or more generally, $m \times n$ rectangle).

There is a well-known formula for the number of domino tilings of an $m \times n$ rectangle:

$$2^{mn/2} \prod_{\substack{1\leq i\leq m \\ 1\leq j\leq n}} \left(\left(\cos\frac{i\pi}{m+1}\right)^2 + \left(\cos\frac{j\pi}{n+1}\right)^2\right)^{1/4},$$

which is derived by analyzing 1-factors in rectangular meshes. (see [4][Chapter 7, Exercise 51] and [6][Exercise 4.29]). We may consider the combinatorial function $\mathrm{SQ}_{m\times n}(k)$ = "the number of different domino tilings of an $m \times n$ rectangle which have exactly k pairwise nonoverlapping subtilings each consisting of a 2×2 square". Our previous study computes $\mathrm{SQ}_{n\times 2}(k), \mathrm{SQ}_{n\times n}(1), \mathrm{SQ}_{n\times n}(2)$ exactly. However, in general, no combinatorial formula is known for this function.

References

[1] S. W. Golomb, "Checkerboards and Polyominoes", American Mathematical Monthly LXI, December 1954, 10, pp. 672 - 682.

[2] S. W. Golomb, "Polyominoes", Princeton Science Library, Princeton University Press, 1994. (Original edition published by Charles Scribner's Sons, 1965.

[3] S. W. Golomb, "Polyominoes which tile rectangles", Journal of Combinatorial Theory, Series A 51, no 1 (1989), pp. 117 - 124.

[4] R. L. Graham, D. E. Knuth, and O. Patashnik, "Concrete Mathematics", Addison-Wesley, 2nd edition, 1994.

[5] B. Grünbaum and G. C. Shephard, "Tilings and Patterns", W. H. Freeman and Company, New York, 1987.

[6] L. Lovasz, "Combinatorial Problems and Exercises", North-Holland, 1979.

[7] N. C. Saldhana, C. Tomei, M. A. Casarin, D. Romualdo, "Spaces of Domino Tilings", Discrete Computational Geometry, 14:207-233 (1995).

[8] W. P. Thurston, "Conway's Tiling Group", American Mathematical Monthly, pp. 757-773, October, 1990.

[9] H. S. Wilf, "Generatingfunctionology", Academic Press, 2nd edition, 1994.

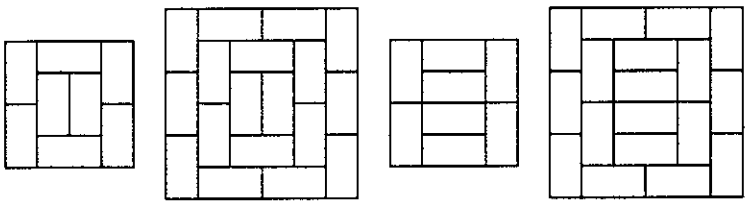


Figure 2: The two leftmost squares depict a 4×4 and a 6×6 tiling with a unique 2×2 square and the two rightmost a 4×4 and a 6×6 tiling with exactly two nonoverlapping 2×2 squares. These figures can be used to generate for each even n an $n \times n$ square with a unique (respectively, exactly two nonoverlapping) 2×2 square(s) by "surrounding" the 4×4 squares with a layer of dominoes, and so on recursively.

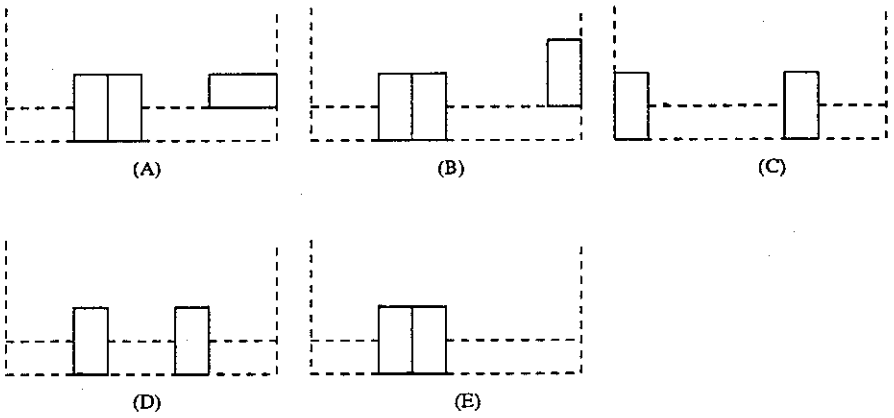


Figure 3: The characterization of domino tilings with one or two nonoverlapping 2×2 squares.

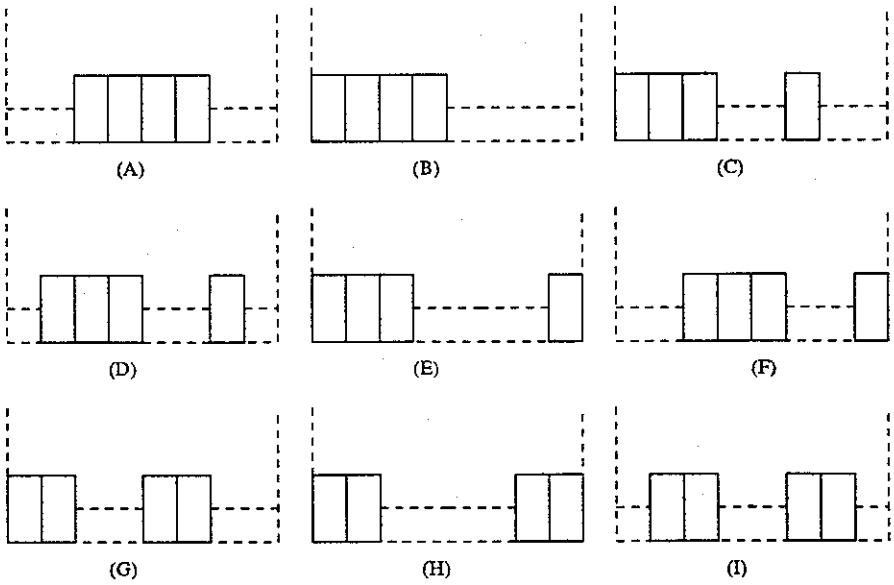


Figure 4: The characterization of domino tilings with exactly two nonoverlapping 2×2 squares.

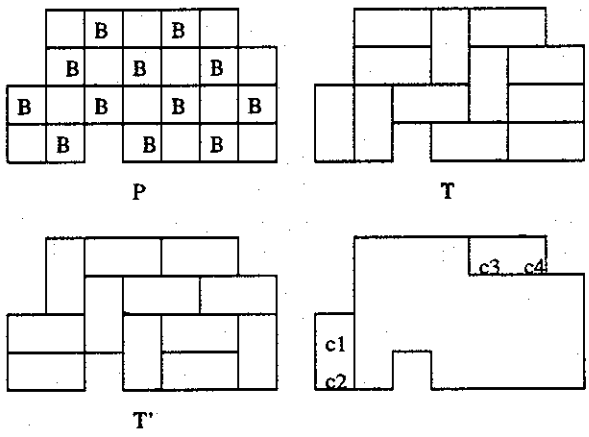


Figure 5: A simple polygon and two different domino tilings T, T'. Black cells are labeled B. These two tilings give rise to the corresponding directed graph (not depicted here). The right bottom picture depicts two boundary dominos of the polygon: the vertical with cells c_1, c_2 and the horizontal with cells c_3, c_4.

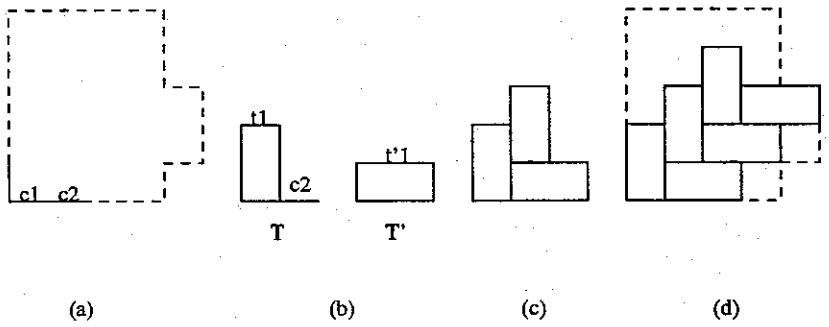


Figure 6: Moving from one tiling to the next by flipping dominoes.

Covering a set of points by two axis-parallel boxes

Sergei Bespamyatnikh* and Michael Segal†

April 25, 1997

Abstract

We present an efficient algorithm for solving the following clustering problem: Given a set S of n points in d-dimensional space, $d \geq 2$, find two axis-parallel boxes b_1 and b_2 that together cover the set S and minimize the expression: $max(\mu(b_1), \mu(b_2))$, where μ is a monotone function of the box, i.e. $b_1 \subseteq b_2$ implies $\mu(b_1) \leq \mu(b_2)$.

1 Introduction

We consider the following min-max two box problem: Given a set S of n points in d-dimensional space, $d \geq 2$, find two axis-parallel boxes b_1 and b_2 that together cover the set S and minimize the expression: $max(\mu(b_1), \mu(b_2))$, where μ is a monotone function of the box, i.e. $b_1 \subseteq b_2$ implies $\mu(b_1) \leq \mu(b_2)$. Examples of box measure μ are the volume of the boxes, their perimeter or the length of their diagonal. This problem is closely related to the rectilinear p-center problems (and in particular to the 2-center problem). The p-center problem is defined as follows. Let $\mathcal{R}$ be the set of compact convex regions with nonempty interior in the plane, where every region $r \in \mathcal{R}$ is assigned a *scaling point* c_r in its interior. For $r \in \mathcal{R}$ and a real number $\lambda \geq 0$, let $r(\lambda)$ be the homeothetic copy of r obtained by scaling r by the factor λ about c_r (i.e., $r(\lambda) = \{c_r + \lambda(a - c_r) | a \in r\}$). Finally, $\mathcal{R}(\lambda) = \{ r(\lambda) | \ r \in \mathcal{R}\}$. The p-center problem for $\mathcal{R}$ looks for

$$\lambda_{\mathcal{R}} = min\{ \ \lambda | \mathcal{R}(\lambda) \text{ is } p\text{-pierceable } \}.$$

We call set $\mathcal{R}$ p-pierceable if there exist a set of p points that intersects each member of $\mathcal{R}$. If $\mathcal{R}$ is a set of translates of a square and the scaling points are the respective centers, then

*Department of Methematics and Mechanics, Ural State University, Ekaterinburg 620083, Russia
†Department of Mathematics and Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel.

we talk about the *rectilinear p-center problem.* If the squares are still axis-parallel but of possible different sizes, then we have the *weighted rectilinear p-center problem*, and if $\mathcal{R}$ is a set of arbitrary axis-parallel rectangles (and the scaling points are also arbitrary), then we face the *general rectilinear p-center problem.*

Results for the above defined problems, for $p = 2$ and $d = 2$, were obtained by [4, 1, 7]. Hershberger and Suri [4] solve the following clustering problem: Given a planar set of points S, a *rectangular measure* μ acting on S and a pair of values μ_1 and μ_2, does there exist a bipartition $S = S_1 \cup S_2$ satisfying $\mu(S_i) \leq \mu_i$ for $i = (1, 2)$? They present an algorithm which solves this problem in $O(n \log n)$ time. Based on this algorithm and using the sorted matrix technique of Frederickson and Johnson [2], Glozman et.al [1] obtained an $O(n \log n)$ time algorithm that solves min-max box problem in the plane. In a very recent paper Sharir and Welzl [7] using LP-type framework and Helly-type results obtained an $O(n)$ expected time algorithm for the general rectilinear 2-center problem, where $d = 2$. The paper of Segal [5] solves the same problem as this paper but only in the case where $d = 3$. The runtime of the algorithm in [5] is $O(n^2 \log n)$.

In this paper we present an efficient algorithm for solving the min-max two box problem in arbitrary dimension $d \geq 2$. The runtime of the algorithm is $O(dn \log n + d\binom{2d}{d} n^{d-1})$. This paper is organized as follows. In Section 2 we present our algorithm for dimension $d \geq 3$. The plane case will be described in full version of the paper. We conclude in Section 3.

2 The algorithm

Given a set $S = \{p_1, \ldots, p_n\}$ of n points in $d, d > 2$, find two axis-parallel boxes b_1 and b_2 that together cover the set S and minimize the expression: $\max(\mu(b_1), \mu(b_2))$, where μ is a monotone function of the box, i.e. $\mu(b_1) \leq \mu(b_2)$ if $b_1 \subseteq b_2$. For simplicity we consider the general case of distinct coordinates, i.e. the projection S onto any coordinate axis is a set of n distinct points. Initially we sort the points of S according to each of the coordinates.

We start with some notations and observations. Given a set of points S, the *bounding box* of S, denoted by $bb(S)$, is the smallest axis-parallel box that contains S. The bounding box of S is determined by $2d$ points, two from each axis i, $i = 1, \ldots, d$: the leftmost point

$l_i(S)$ of S, and the rightmost point $r_i(S)$. We call these points the *determinators* of S. For a box $b = [l_1, r_1] \times \ldots \times [l_d, r_d]$ we also call l_i, r_i the determinators of b. For a point p, the values $x(p), y(p)$ denote the coordinates of p in the first and the second axes respectively.

The main idea of the algorithm is the reduction of the dimension. We assume that the dimension d is greater than two and reduce it to two. Let the boxes b_1 and b_2 be the solution of the min-max two box problem. Then there is a box which has at least d determinators coinciding with the corresponding determinators of S. We can assume that it is b_1. It is easy to see that the boxes b_1 and b_2 that solve the min-max problem are the ones that solve the following problem.

Problem P1. Given a set S of n points and d determinators of the box b_1, and denote by $b_2 = bb(S \setminus b_1)$. Find the remaining d determinators of b_1 such that the expression $\max(\mu(b_1), \mu(b_2))$ is minimized.

There are $\binom{2d}{d}$ problems P1. We solve all these problems. The solution of the min-max two box problem can be chosen from $\binom{2d}{d}$ pairs of boxes b_1 and b_2.

Now we show how to solve problem P1. Since the dimension d is greater than 2, there are at least two determinators of b_1 which are not equal to the determinators of S in the corresponding axes. Assume wlog that they are $r_1(S)$ and $r_2(S)$. We still have to find the other $d-2$ determinators of b_1. We can assume that they are determined by other points of S (for example the determinator $l_i(b_1)$ is determined by point $p = (p_1, \ldots, p_d) \in S$ if $l_i(b_1) = p_i$). Let us consider all the $(d-2)$-tuples of points S. The algorithm scans all these tuples and finds the combination that attains the required minimum.

It is clear that the number of tuples is n^{d-2}. Note that some tuples cannot give the solution because the determinators of b_1 have to satisfy $l_i < r_i, i = 1, \ldots, d$.

The 2-dimensional problem can be now formulated as

Problem P2. Given a set S of n points and $2d-2$ determinators of the box b_1 $l_1, \ldots, l_d, r_3, \ldots, r_d$. Find the two determinators r_1 and r_2 of the box b_1 that minimize the expression

$$\max(\mu(b_1), \mu(bb(S \setminus b_1))).$$

We show that this problem can be solved in $O(dn)$ time. The determinators r_1 and r_2 can be chosen from the sets $\{x(p_1), \ldots, x(p_n)\}$ and $\{y(p_1), \ldots, y(p_n)\}$ respectively. Let

$b(x,y)$ denote the box $[l_1,x]\times[l_2,y]\times[l_3,r_3]\times\ldots\times[l_d,r_d]$, where $x\in\{x(p_1),\ldots,x(p_n)\}$ and $y\in\{y(p_1),\ldots,y(p_n)\}$. Let S_1 denote the set of points of S in the box $b(x,y)$ and $S_2=S\setminus S_1$. For each $x\in\{x(p_1),\ldots,x(p_n)\}$, our algorithm finds the largest $y\in\{y(p_1),\ldots,y(p_n)\}$ such that $\mu(b(x,y))\leq\mu(bb(S\setminus b(x,y)))$. Denote it by $Y_1(x)$. We observe that $Y_1(x)$ is *monotone*.

Observation 2.1 *$Y_1(x)$ is nonincreasing function.*

For each $x\in\{x(p_1),\ldots,x(p_n)\}$ our algorithm (for Problem P2) finds the smallest $y\in\{y(p_1),\ldots,y(p_n)\}$ such that $\mu(b(x,y))>\mu(bb(S\setminus b(x,y)))$. Denote it by $Y_2(x)$.

Observation 2.2 *There exists a solution of problem P2 such that $r_2=Y_1(r_1)$ or $r_2=Y_2(r_1)$.*

For each $x\in\{x(p_1),\ldots,x(p_n)\}$ and $y\in\{Y_1(x),Y_2(x)\}$, we compute $\max(\mu(b(x,y)),\mu(bb(S\setminus b(x,y))))$. Then we compute minimum value of these numbers. It gives the solution of problem P2.

Now we explain how to achieve $O(dn)$ running time. Let us consider the moment when we have computed $Y_1(x)$ and $Y_2(x)$ for some x. Using the fact that the points of S are sorted sperately in each of the coordinates we get the next value $x'>x$ in this order. For the point p of S with x-coordinate x', we perform the following operations.

First we check whether p can lie in b_1. If p cannot lie in b_1 (the i-th coordinate of p is less than l_i for some i, or the i-th coordinate of p is greater than r_i, for some $i>2$) then p remains in S_2 and we pass it. Otherwise compare $y(p)$ and $Y_1(x)$. If $y(p)>Y_1(x)$ then p remains in S_2 (by monotonicity of Y_1) and we pass it. Otherwise we delete p from S_2 and insert it into S_1. For the determinators $r_1=x'$ and $r_2=Y_1(x)$ of the box b_1 compute $\mu_1=\mu(b_1)$ and $\mu_2=\mu(bb(S_2))$. There are two possible cases.

Case 1: $\mu_1\leq\mu_2$. In this case $Y_1(x')=Y_1(x)$ and $Y_2(x')=Y_2(x)$ by monotonicity of Y_1. It should be noted that we do not need to compute the rectangular measure μ for the pair x' and $Y_2(x)$ since it is greater or equal than the measure of the boxes b_1 and b_2 determined by the pair x and $Y_2(x)$. We only have to compute the rectangular measure $\max(\mu_1,\mu_2)$ for the pair x' and $Y_1(x)$ and update the current solution if its measure is greater than $\max(\mu_1,\mu_2)$.

Case 2: $\mu_1>\mu_2$. In this case $Y_1(x')<Y_1(x)$ and $Y_2(x')<Y_2(x)$. Let $y=Y_1(x)$. In order to find $Y_1(x')$ and $Y_2(x')$ we perform the following operations as long as $\mu_1>\mu_2$.

Proof. We spend $O(dn \log n$ time in sorting S according to all its coordinates. As was pointed above, we solve $\binom{2d}{d}$ problems P1 in order to obtain a solution for the min-max two box problem. Each P1 problem is solved by considering all the $(d-2)$-tuples of points S. There are n^{d-2} such tuples. For each such a tuple related to the problem P2 which we solve in $O(dn)$ time. ∎

3 Conclusions

In this paper we present an efficient algorithm for solving min-max two box problem. The efficiency of the algorithm is based on the monotonicity of the evaluated function in the problem. It would be interesting to find some connection between this problem and the problems considered by Jaromczyk and Kowaluk [3] and Segal [6]. Computing lower bounds for this problem and problems in [3, 6] are still open questions.

References

[1] A. Glozman, K. Kedem, G. Shpitalnik "On some geometric selection and optimization problems via sorted matrices", *WADS'95, 1995, pp.26-37.*

[2] G. N. Frederickson, D. B. Johnson "The complexity of selection and ranking in $X+Y$ and matrices with sorted columns", *J. Comput. Syst. Sci.*, 24 (1982), pp. 197-208.

[3] J. W. Jaromczyk, M. Kowaluk *"Orientation independent covering of point sets in R^2 with pairs of rectangles or optimal squares"*, European Workshop on Comp. Geometry, Muenster, Germany, 1996.

[4] J. Heshberger, S. Suri "Finding tailored partitions", *J. Algorithms*, 12 (1991), pp. 431-463.

[5] M. Segal unpublished manuscript.

[6] M. Segal "Covering the set of points by two shapes under different constraints", in preparation.

[7] M. Sharir, E. Welzl "Rectilinear and polygonal p-piercing and p-center problems", In *Proc. 12th ACM Symp. on Computational Geometry*, 1996.

- If there is a point $p \in S_1$ with $y(p) = y$ then move the point p from S_1 to S_2.
- Using the sorted order of S points according to the y-coordinate we move to the next point whose y coordinate is smaller than the one we currently have.
- Compute $new\_\mu_1 = \mu(b(x', y))$ and $new\_\mu_2 = \mu(bb(S_2))$. Update μ_1 to be new_μ_1 and μ_2 to be new_μ_2 if $\max(\mu_1, \mu_2) > \max(new\_\mu_1, new\_\mu_2)$.

After these operations we have $Y_1(x') = y$ and $Y_2(x')$ is the previous value of y. Now we finish processing the coordinate x'.

We start the algorithm at a point x that is less than the x-coordinate of all the points of S, for example $x = \min\{x(p_1), \ldots, x(p_n)\} - 1$. We set $Y_1(x) = \min\{y(p_1), \ldots, y(p_n)\}$. Initially we have $S_1 = \emptyset$, $\mu_1 = 0$, $S_2 = S$ and $\mu_2 = \mu(bb(S))$, Y_2 can have any value because it is used only after S_1 becomes non-empty.

Ignoring the time spent on computing μ_1 and μ_2 the algorithm takes $O(dn)$ time. It remains to show how to compute μ_2.

Computing μ_2

Recall that $\mu_2 = \mu(S_2)$. The set S_2 undergoes both insertions and deletions of points throughout the algorithm, but one point can be deleted from and inserted to S_2 only once. Initially $S_1 = \emptyset$ and $S_2 = S$. We partition S_2 into two subsets S_2' and S_2'' defined below:

- S_2' contains the points that were not deleted from S_2
- $S_2'' = S_2 \setminus S_2'$.

The set S_2' is updated only by deletion of points. The set S_2'' is updated only by insertion of points. It is easy to maintain the box $bb(S_2'')$. Using presorting we can maintain the box $bb(S_2')$ in $O(d)$ time. The box $bb(S_2)$ can be computed in $O(d)$ time using the boxes $bb(S_2')$ and $bb(S_2'')$.

We summarize with the following theorem

Theorem 2.3 *The min-max two box problem in d dimensional space, $d \geq 2$, can be solved in time $O(dn \log n + d\binom{2d}{d} n^{d-1})$.*

Encoding a triangulation as a permutation of its point set

Markus O. Denny
FB 14 Informatik
Universität des Saarlandes
D-66123 Saarbrücken
FAX:+49-681-302-5576
mdenny@cs.uni-sb.de

Christian A. Sohler
FB 14 Informatik
Universität des Saarlandes
D-66123 Saarbrücken
FAX:+49-681-302-5576
csohler@cs.uni-sb.de

Abstract

We present algorithms that given a straight edge triangulation of n points in the plane encode a triangulation as a permutation of the points. A first algorithm, of rather theoretical interest, realizes the encoding and decoding in $O(n)$ time. We also present a more practical algorithm consuming $O(n \log n)$ time. As a byproduct of this work we get a new upper bound on the number of triangulations of planar point sets of at most $2^{8.2n+O(\log n)}$.

1 Introduction

It is well known that the number of triangulations of n points in the plane is bounded from above by $2^{O(n)}$ [1]. Since there are approximately $2^{n \ln n}$ different permutations of a point set of size n, it is obvious that it is possible to encode a triangulation as a permutation of the input data.

In the case of *canonical* geometric structures related to Delaunay triangulations Snoeyink and van Kreveld [2] already introduced a linear time algorithm and listed a number of advantages. The graphs are still readable by many other applications and transmission overhead is reduced.

However our approach is a more general one. We show that it is possible to efficiently encode *any* straight edge triangulation as a permutation of its point set. Inspired by the planar point location method of Kirkpatrick [3] we present an algorithm that encodes and decodes every triangulation of sufficiently large size in $O(n)$ time. Since this algorithm only works for relatively large triangulations we also present a more practical approach based on the sweep line paradigm which encodes and decodes a triangulation in $O(n \log n)$ time. From this algorithm we also obtain a new upper bound on the number of straight line triangulations.

Let T denote the triangulation under consideration given as a set V of n points together with the set E of the corresponding edges. We assume w.l.o.g. that it is always possible to add two points to the triangulation such that the new convex hull forms a triangle around the original triangulation (for instance the virtual points $(x_{max}+1, \infty)$ and $(x_{max}+1, -\infty)$ will suffice our conditions where x_{max} denotes the maximum x-coordinate of all points in V).

2 Encoding a triangulation as a permutation

At first we provide generic algorithms for encoding and decoding.

Each generic algorithm consists of $O(\log n)$ phases. In the first part of each phase of the encoding algorithm a constant fraction of points with low degree is removed from the current triangulation and put into an initially empty set W. (For the sake of simplicity we never remove any of the 3 points of the convex hull of V.) Next we retriangulate the created holes retrieving a new triangulation T'. In the second part we encode the information needed to reconstruct the original triangulation from T' into a permutation of the points of W.

The decoding algorithm works in a reverse manner. Having read a bunch of points we insert them in the right place into the current triangulation T' meanwhile computing their permutation. With the corresponding information we can now determine the appearance of the former triangulation and finally restore the original triangulation.

PART 1 OF *generic* ENCODING:

- pick a constant fraction W of n points with low degree
- remove W from the triangulation
- retriangulate

PART 2:

- compute and output the permutation of W

PART 1 OF *generic* DECODING:

- read the next permutation of points
- find the right places to insert them
- regain the information from the permutation

PART 2:

- delete the superfluous edges and retriangulate

3 A linear time algorithm

3.1 The algorithm

We now specify how a phase of the linear time algorithm looks like:

PART 1 OF ENCODING:

- compute a maximal independent set I of vertices with degree at most 6
- remove I from the triangulation
- retriangulate

PART 2:

- visit all triangles of the new triangulation in a canonical way (DFS, BFS) to order I (cf. [2])
- partition I into sets A_j of constant size
- store the information needed for the reconstruction by permutating each of these A_j
- Output A_j

PART 1 OF DECODING:

- read I from the input
- visit all triangles in the same manner as in part 2 of ENCODING to order I and to determine the enclosing triangle for each $v \in I$ (see figure 1.a)
- obtain the information needed to reconstruct the triangulation from the permutation

PART 2:

- use the information gained in part 1 to restore the triangulation (see figure 1.b-d)

3.2 Correctness

First of all we show that we can compute the size of I in each phase of DECODING and also the number of points that have been removed in part 2 of ENCODING. But this proves to be easy since we know n and we can simply restrict the size of the maximal independent set I in ENCODING to a constant fraction of V and thus immediately determine the sizes of I and the A_j's. W.l.o.g. we can assume an order on the input to ensure the same enumeration of the triangles for both ENCODING and DECODING induced by DFS or BFS respectively (e.g. ccw-ordering of the edges).

It is easy to verify that the algorithm will always terminate because in each phase the number of points is always reduced by a constant fraction.

3.3 Running time

To obtain a maximal independent set I we use the greedy algorithm presented by Snoeyink and van Kreveld [2] with the difference that our maximal allowed degree d is 6. The linear time algorithm removes at first all points with degree at least $d+1$. Then the point with lowest degree is put into the set I and its neighbours are deleted. This procedure is repeated until no point remains. Analyzing the behaviour of the algorithm leads to a linear program the optimum of which turns out to be $\frac{n}{10}$ for the minimal size of I. The deletion of the points and the retriangulation can be done on the fly. This gives a running time of $O(n)$ for the first phase of ENCODING.

In the second phase we can establish an ordering of the points in linear time since we know which new triangle contains the deleted point (cf. [2]). Since each A_j has constant size the running time for both phases is $O(n)$. Thus the overall running time for ENCODING as well as DECODING is $O(n)$ since the DECODING is essentially the reverse of the ENCODING.

3.4 How reconstruction works

Consider a triangulation T. If a point p with degree d is removed the originated hole forms a polygon. The triangulation of this polygon is dual to a tree b with the following properties (see figure 2). The root of b (representing the p-enclosing triangle) has degree at most 3, the inner nodes have degree at most 2 and the sum of all edges is $d-3$. The number $h(d)$ of such trees is given by:
$h(d) = \sum_{i_j} \binom{3}{i_1}\binom{2i_1}{i_2}\cdots\binom{2i_{d-4}}{i_{d-3}}$ with $\sum_j i_j = d-3$.

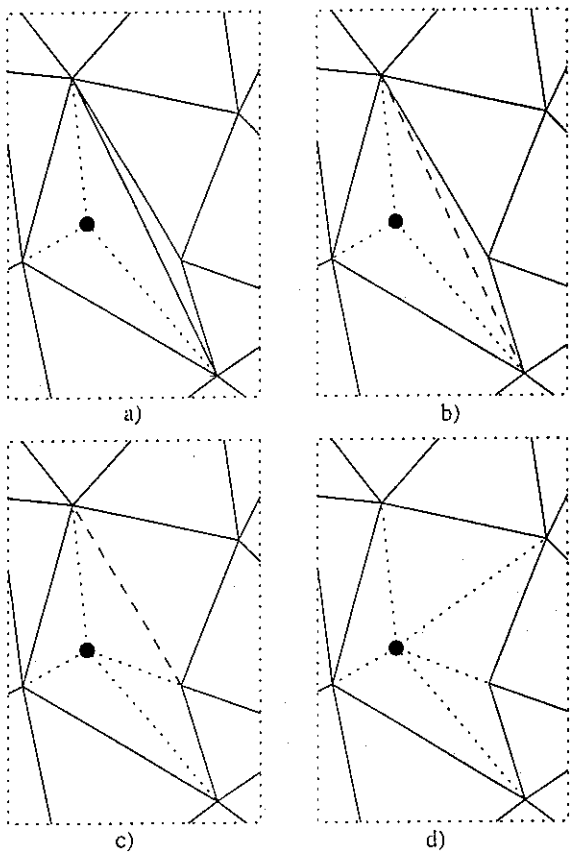


Figure 1: Reconstruction of a triangulation: a) determine the enclosing triangle and draw the first three edges (dotted lines) b)-d) According to the dual tree (see figure 2) choose the next edges to be flipped (dashed lines)

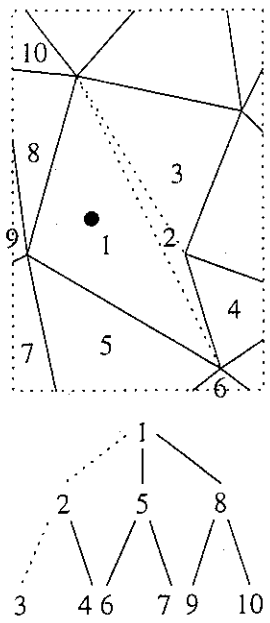


Figure 2: The dual tree (dotted lines) which describes the retriangulation to be performed

3.5 Analysis

Since we limited the degree of each removed vertex to 6, we get exactly $\Sigma_{3\leq i\leq 6} h(i) = 41$ possibilities to insert a point. Thus we need $\lceil n \log_2(41) \rceil$ bits for the encoding. The size of the independent set has to be large enough to encode this information. Since $109! > 41^{109}$ it suffices to have at least 109 points to build a block. Therefore the triangulation must have at least $10 * 109 = 1090$ points at the beginning of a phase. If during the course of the algorithm we arrive at a triangulation with fewer points, we have to use another algorithm to encode this triangulation, e.g. the one stated below.

4 An $O(n \log n)$ time algorithm

4.1 A sweep line algorithm

The algorithm we present in this section is based on the sweep line paradigm. We show that it is possible to remove a much bigger fraction of vertices with minimum degree 6 if we do not need an independent set. In a precomputing step we sort the point set of the triangulation. Now we specify how a phase of the second algorithm looks like.

PART 1 OF ENCODING:

- initialize the set of removed nodes I with $\emptyset$
- sweep over the triangulation from left to right and stop at each vertex v
- if $deg(v) \leq 6$ then remove v from the triangulation, retriangulate and add v to I

PART 2 OF ENCODING:

- store the information needed for reconstruction by permutating I
- output I

PART 1 OF DECODING:

- read I from input
- sort I lexicographically
- sweep over the triangulation from right to left and stop at each $v \in I$
- search the enclosing triangle
- retrieve the information needed to reconstruct the triangulation from the permutation

PART 2 OF DECODING:

- use the information gained in Part 1 to insert v

4.2 Correctness

We sweep from right to left in the DECODING algorithm. Thus DECODING does exactly the same as ENCODING but in reversed direction. We use common sweep line techniques to maintain an order of the triangles that intersect the sweep line. Using this order, we can easily find the enclosing triangle. All other steps are either already discussed or trivial.

4.3 Running time

We have to spend $O(n \log n)$ time for sorting. All steps of part one and two need a constant amount of time except the search for the enclosing triangle which needs $O(\log n)$ time for each vertex and the sorting of I which needs $O(n \log n)$ time. To maintain the lexicographical order during DECODING we also need $O(\log n)$ time for each inserted point. Overall, we have a running time of $O(n \log n)$ for ENCODING and DECODING.

4.4 Analysis

As one can easily see the degrees of the three vertices of the convex hull sum up to at least 10 (if T consists of more than 4 vertices). Let n denote $|V|$, n' the vertices that remain after the sweep line phase (excluding the 3 points of the convex hull), and r the number of removed vertices. Each point at the left of the sweep line must have at first a minimal degree of 7 since it was not removed. For each removed vertex the total degree of all vertices on the left of the sweep line decreases at most by 2 (see figure 3).

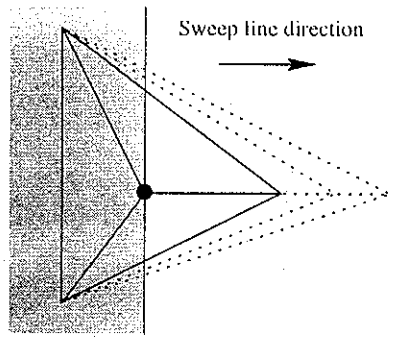


Figure 3: a worst case situation

At the end of each phase the remaining $n'+3$ points form a complete triangulation. Thus the following inequalities describing the sum of degrees hold:

$$\begin{aligned} 7n' - 2r + 10 &\leq 6(n'+3) - 12, \text{ and hence} \\ n' &\leq 2r - 4 \end{aligned}$$

By definition n' and r sum up to $n-3$. That yields the desired result of $r \geq \lfloor \frac{n}{3} \rfloor$ immediately. From the analysis of the first algorithm it follows that at least 109 points are necessary to encode the information for reconstruction. This makes it possible to execute a phase of the second algorithm for triangulations with at least 326 points. As soon as fewer than 326 points remain we cannot store enough information in the permutation. Provided a sufficiently large triangulation we encode this information in previously computed permutations. Overall, we have to store $\lceil n \log_2 41 \rceil$ bits. Thus we need at least approx. 750 points to store all the information.

4.5 A more practical point of view

If we had two more bits for each vertex we could encode any triangulation independent of the size of its point set. A practical way to get these two bits is to translate all points to reside in the upper right quadrant. The two additional bits for each point can now be stored as the sign of the coordinates. We now have to encode $\lceil n((\log_2 41) - 2) \rceil$ bits resulting in a minimal size of approx. 190 points for the triangulation.

5 New upper bounds

5.1 Idea

To derive an upper bound on the number of triangulations we slightly change the algorithm presented in the previous section. Instead of coding the information needed for reconstruction into a permutation we now encode it as a binary string $\omega(T)$. If every possible encoding string has length $f(n)$, then clearly V cannot have more than $2^{f(n)}$ different triangulations.

When encoding with a binary string, we loose the implicit information about the insertion order that was stored in the permutation. Thus we need the additional information which points were removed in each phase of the algorithm.

5.2 Analysis

For the sake of simplicity we assume that the output is ordered such that the first part of $\omega(T)$ represents the encoding for the local changes of each vertex at the time of its deletion. Let $\varsigma(E)$ denote this part of $\omega(T)$. The rest of the binary string $\sigma(V)$ contains

the information which vertices were removed at each part. Thus we have:

$\omega(T) = \varsigma(E)\sigma(V)$

$\sigma(V') = \varsigma(W)\sigma(V' \setminus W)$

If $|V'| = 3$ there is only one possible triangulation and we encode it as the empty string. To encode W, we proceed as follows (cf. [4]). As a first step we use the characteristic vector representation for $\varsigma(W)$. That's a binary string s of length $|V'|$ with $s_i = 1$iff $v_i \in V'$. This can then be slightly improved by using a better encoding for $\varsigma(W)$. Since W is a subset of size exactly $\lfloor \frac{n}{3} \rfloor$ there are only $b(n) = \binom{n}{\lceil \frac{2n}{3} \rceil}$ choices for W. Hence W can be encoded by a string of length $\log_2 b(n)$. Since $\binom{n}{\alpha} \leq 2^{nH(\alpha)}$, where $H(x) = -x \log_2 x - (1-x)\log_2(1-x)$ we get that W can be encoded by a binary string of length $H(1/3)n \leq 0.9183n$ The impact of this fact will be stated in the following theorem.

Theorem

The number of triangulations is bounded from above by $2^{f(n+2)}$ *with* $f(n) \leq (\log_2(41) + 3\,H(\frac{1}{3}))n \leq 8.12\,n + O(\log n)$.

Taking into account that every crossing free subgraph G of K^n can be encoded in the same way as a triangulation in combination with a binary vector of size $3n$ coding the added edges, the following corollary is an direct consequence of the above Theorem.

Corollary 1:

The number of undirected crossing free subgraphs of the complete graph K^n *is bounded from above by* $2^{f(n+2)+3n}$.

We can also derive an upper bound on the number of directed crossing free subgraphs since every edge of the triangulation can either be deleted, replaced by a single one-directional arc or by two arcs for each direction.

Corollary 2:

The number of directed crossing free subgraphs of the complete graph K^n *is bounded from above by* $2^{f(n+2)+6n}$.

6 Acknowledgements

We would like to thank Raimund Seidel for the inspiring discussions and his support of this work.

References

[1] M. Atjai, V. Chvàtal, M. Newborn, E. Szemerédi, *Crossing-free Subgraphs,* Annals of Discrete Math. 12 (1982), 9-12

[2] J. Snoeyink, M. van Kreveld, *Good orders for incremental (re)constructions,* 13th Annual ACM Symposium on Computational Geometry 97

[3] D. Kirkpatrick, *Optimal search in planar subdivisions,* SIAM Journal on Computing, 12:28-35, 1983

[4] R. Seidel, *On the number of triangulations of planar point sets,* to appear in Combinatorica

Recognizing the Trivial Knot by Planar Diagrams.

Wolfgang Haken
University of Illinois

A solution of the so-called Knot Problem, i.e., an algorithm to recognize the trivial knot was published in 1961, [1]. This algorithm entirely avoided the use of "higher" algebra and proceeded by elementary geometric topology and integer programming. The method became known as the method of normal-surfaces which can be used to algorithmically determine incompressible surfaces in compact 3-manifolds with or without boundaries.

Although the equations and inequalities to be solved are all linear the algorithm is extremely time consuming. This is since it requires to find all non-negative integral fundamental solutions of a linear system - and those are exponentially many.

Attempts to reduce the time demand of the algorithm have been at least partially successful:

1) The complement of the given (slightly thickened) knot is a compact 3-dimensional manifold whose boundary is a torus. This 3-manifold can be presented by a so-called normal-partition (very similar to a handle partition). The normal-partition can again be conveniently presented by a planar diagram which is a cell-partition of the plane with labeled elements. A polynomial time-demand method to reduce the so-called KF-complexity of the normal-partition was developed. This method uses simplification procedures of the planar diagram and it can be applied to normal-partitions of arbitrary (compact) 3-dimensional manifolds with or without boundaries.

2) The reduction method (1) alone turned out to be a very effective partial algorithm: It can be carried out for given knot projections of more than thousand overcrossings on a home computer and if the given knot is trivial then it usually recognizes this by reducing the KF-complexity to zero. But it does not appear to be reasonable to speculate that this method always recognizes the trivial knot - although so far no counter-examples could be found.

3) Thus, in order to obtain an algorithm of which one can prove that it is both polynomial and complete, we must find an effective procedure for treating the case that the reduction method (1) reduces the KF-complexity to a value greater than zero. But this procedure must avoid the necessity to find all the fundamental solutions of a linear system and should be restricted to the tools of linear programming.

A promising attempt to find such a procedure is as follows:

3.a) In the given normal-partition, one can construct, in polynomial time, a so-called normal-surface, N1, which is orientable, non-separating, and has exactly one boundary curve. That curve is then parallel to the originally given knot.

3.b) If N1 happens to be a disk then it exhibits the given knot as trivial and the procedure halts.

Otherwise, N1 is a surface of genus > 0. Then we have to find out whether N1 is compressible (i.e., its genus can be reduced by replacing an annulus in N1 by two disks in the complement of N1) or incompressible. (That test is described in 3.c below.)

If N1 is incompressible then its genus is the genus of the knot, i.e. the knot is non-trivial and the procedure halts.

Otherwise, we replace N1 by a surface of smaller genus and we repeat (3.b) with the new N1.

3.c) The surface N1 cuts the normal-partition up, producing another normal-partition with a larger number of elements but fortunately, with smaller KF-complexity. Thus the new normal-partition can then be simplified, by the method (1), so that it has finally fewer elements than the old one.

If the KF-complexity of the new normal-partition is 0 then we have a normal-partition of a handlebody (the boundary of which contains the two sides of N1). Then we use the obvious meridian disks for cutting the handlebody up into a 3-cell. The boundary of that 3-cell is then a planar diagram which shows the sides of the meridian disks, parts of the sides of N1, and parts of the boundary torus of the knot complement. That diagram permits an easy test whether or not N1 is compressible.

Otherwise, if the new normal-partition still has KF-complexity > 0, we construct a second normal-surface, N2, and use it for further cutting up the normal-partition (again lowering the KF-complexity), etc, etc, until we obtain a hierarchy of orientable normal-surfaces, N1, N2, ..., and meridian disks, which cut the knot complement up into one 3-cell. Then the boundary diagram of that 3-cell permits to determine whether or not all the surfaces in the hierarchy are incompressible (and boundary-incompressible). In the positive case, the knot is not trivial and the procedure halts. In the negative case, one can replace one of those hierarchy surfaces which are compressible (or boundary-compressible) by one of lower genus and use this to construct an improved hierarchy. - Only a limited number of such hierarchy improvements can occur. But the question still open at this time is: Can we refine the procedure so far that the limit for the number of hierarchy improvements can be proved to be polynomial (in the number of overcrossings of the given knot projection)?

References

[1] W. Haken, *Theorie der Normalflaechen,* Acta Mathematica, 1961.

A Note on the Tree Graph of a Set of Points in the Plane

Extended Abstract

Eduardo Rivera-Campo

Instituto de Matemáticas, Universidad Nacional Autónoma de México[1]

Virginia Urrutia-Galicia

Departamento de Matemáticas, Universidad Autónoma Metropolitana-Iztapalapa

1.- Introduction

The *tree graph* of an abstract graph G is the graph $T(G)$ that has a vertex for each spanning tree of G and an edge joining two trees S and R whenever R is obtained from S by replacing an edge s in S with an edge r in R. R. Cummins proved in [2] (see also [3]) that if G is connected and has more two vertices, then $T(G)$ contains a Hamilton cycle, that is a cycle which includes every vertex of G.

Let P be a set of n points in general position in the Euclidean plane E^2. A *geometric graph* with vertex set P is a graph G, drawn in E^2 in such a way that every edge is a straight-line segment with ends in P. A *spanning tree* of P is a spanning tree of the complete geometric graph $K(P)$ with vertex set P.

The *tree graph* of a set P of n points in the plane is the graph $T(P)$ with a vertex for each non-selfintersecting spanning tree of P; two trees are adjacent in $T(P)$ if and only if they are adjacent in the tree graph of the complete graph K_n with vertex set P. In this note we prove that $T(P)$ is always connected and give bounds for its diameter and for its minimum and maximum degrees. For graph theoretical concepts we refer to [1].

2.- Connectivity of $T(P)$

For any vertices u and v of a connected graph G, we denote by $d_G(u,v)$ the distance between u and v in G, that is the minimum number of edges in a path joining u and v in G. In this section we show that if P contains at least two points, then there is a non-selfintersecting spanning tree of P which is at distance at most $n-2$ in $T(P)$ from any other non-selfintersecting spanning tree of P.

---

[1] On sabbatical leave from Universidad Autónoma Metropolitana-Iztapalapa

For the remaining of this note we identify the non-selfintersecting spanning trees of P with the plane trees of the complete geometric graph $K(P)$. The following lemma is used in the proof of the main result.; it is presented here without proof.

Lemma 2.1.- *Let L be a collection of straight line segments in the plane with pairwise disjoint relative interiors and such that their endpoints are in general position. For any point v outside the convex clousure of L, there is a line segment xy in L such that the triangle xvy does not intersect the relative interior of any segment in L other than xy.* ♦

Theorem 2.2.- *Let $P=\{v_0,v_1,\ldots,v_{n-1}\}$ be a set of $n\geq 2$ points in general position in the plane with v_0 in the convex hull of P. Let S be the plane spanning tree of $K(P)$ with edge set given by $E(S)=\{v_0v_k : k=1,2,\ldots n-1\}$. For any plane spanning tree Q of $K(P)$ there is a path in $T(P)$, joining Q and S and with length at most $|E(Q)\setminus E(S)|$.*

Proof.- If $|E(Q)\setminus E(S)|=0$, then $Q=S$ and the result holds. We assume that the result is valid whenever R is a plane spanning tree of P with $|E(R)\setminus E(S)|=m$ and let Q be a plane spanning tree of $K(P)$ with $|E(Q)\setminus E(S)|=m+1$.

Let $L=E(Q)\setminus E(S)$; since v_0 is outside the convex clousure of L, then by Lemma 2.1, there is an edge xy of Q not in S such that the triangle xv_0y does not meet the relative interior of any edge in $E(Q)\setminus E(S)$ other than xy. Since Q is a plane spanning tree of $K(P)$, then at most one of the edges v_0x and v_0y is an edge of Q; without loss of generality we assume $v_0y\notin E(Q)$ and that xy is in the unique path in Q joining v_0 and y. Since v_0y does not intersect the relative interior of any edge of Q, then $R=(Q+v_0y)-xy$ is a plane spanning tree of P adjacent to Q in $T(P)$.

Since $v_0y\in E(S)\setminus E(Q)$, then $|E(R)\setminus E(S)|=m$. By induction there is a path in $T(P)$, with length at most m, joining R and S. Since R is adjacent to Q in $T(P)$, then $T(P)$ contains a path, joining Q and S, with length at most $m+1$. ♦

Corollary 2.3.- *If P is a set of $n\geq 2$ points in general position in the plane, then $d_{T(P)}(Q,R)\leq 2(n-2)$ for any plane spanning trees Q and R of $K(P)$.* ♦

Let $P=\{u_1,u_2,\ldots,u_m,v_1,v_2,\ldots,v_m\}$ be the set of vertices of a regular $2m$-gon ordered as usual. Let Q and R be the plane spanning trees of $K(P)$ with edge sets $E(Q)=\{u_1v_1,\ldots,u_1v_m,v_1u_2,\ldots,v_1u_m\}$ and $E(R)=\{u_mv_1,\ldots,u_mv_m,v_mu_1,\ldots,v_mu_{m-1}\}$. We claim that every path joining Q and R in $T(P)$ has length at least $3m-4$.

3.- Minimum degree and maximum degree of $T(P)$

For a graph G and a vertex u of G, we denote by $d_G(u)$ the degree of u in G and by $\delta(G)$ and $\Delta(G)$ the minimum and maximum degree of G, respectively. In this section we give a lower bound for $\delta(T(P))$ and an upper bound for $\Delta(T(P))$.

Let S be a plane spanning tree of $K(P)$. For every edge $e \in E(K(P)) \setminus E(S)$, there is a unique cycle S_e of $K(P)$ contained in $S+e$; clearly $(S+e)-x$ is a spanning tree of $K(P)$ for each edge $x \in S_e$. If $(S+e)-x$ is a plane tree and $e \neq x$, then $(S+e)-x$ is adjacent to S in $T(P)$.

Theorem 3.1 $\delta(T(P)) \geq 2(n-2)$ *for any set* P *of* $n \geq 2$ *points in general position in the plane.*

Proof.- Let S be a plane spanning tree of $K(P)$ and D be a triangulation of P that includes every edge of S. Let $e \in E(D) \setminus E(S)$; since $S+e$ is a plane subgraph of $K(P)$, then for each edge $e \neq x \in S_e$, $(S+e)-x$ is a plane spanning tree of $K(P)$, adjacent to S in $T(P)$; therefore

$$\begin{aligned} d_{T(P)}(S) &\geq \sum_{e \in E(D) \setminus E(S)} |E(S_e)| - 1 \\ &\geq 2\left(|E(D)| - |E(S)|\right) \\ &\geq 2\left((2n-3)-(n-1)\right) \\ &= 2(n-2) \end{aligned}$$

♦

Let $P = \{v_0, v_1, \ldots, v_{n-1}\}$ be the set of vertices of a convex n-gon , ordered as usual, and let R be the plane spanning tree of $K(P)$ with edge set given by $E(R) = \{v_0 v_k : k = 1, 2, \ldots n-1\}$. We claim that R is adjacent to exactly $2(n-2)$ trees in $T(P)$. This shows that the bound in Theorem 3.1 is tight. ♦

An upper bound for the maximum degree in $T(P)$ is given here without proof.

Theorem 3.2.- $\Delta(T(P)) \leq \dfrac{(n-1)(n-2)(n+3)}{6}$ *for any set* P of $n \geq 2$ *points in general position in the plane.* ♦

Let $P = \{v_0, v_1, \ldots, v_{n-1}\}$ be the set of vertices of a convex n-gon, ordered as usual, and let S be the plane spanning tree of $K(P)$ with edge set $E(S) = \{v_0 v_1, v_1 v_2, \ldots, v_{n-2} v_{n-1}\}$. In this case $S+e$ is a plane subgraph of $K(P)$ for every edge $e \in E(K(P)) \setminus E(S)$. We claim that the tree S is adjacent to exactly $\dfrac{(n-1)(n-2)(n+3)}{6}$ trees in $T(P)$. This shows that the bound in Theorem 3.2 is also tight. ♦

4.- Final remarks

Let $T^*(P)$ be the spanning subgraph of $T(P)$ in which two trees are adjacent if and only if one is obtained from the other by interchanging two edges with disjoint relative interiors. All results and examples in this note remain valid for $T^*(P)$.

Acknowledgments

We thank Jorge Urrutia for his suggestions and for providing us with the example in section 2.

References

[1] J. A. Bondy, U. S. R. Murty, *Graph Theory with Applications*, North Holland, 1981.

[2] R. L. Cummins, Hamilton Circuits in Tree Graphs, *IEEE Trans. on Circuit Theory*, **CT-13** (1966), 82-90.

[3] H. Shank, A Note on Hamilton Circuits in Tree Graphs, *IEEE Trans. on Circuit Theory*, **CT-15** (1968), 86-.

A straight-line embedding of two or more rooted trees in the plane

M. Kano
Department of Computer and Information Sciences
Ibaraki University, Hitachi 316 Japan
e-mail: kano@cis.ibaraki.ac.jp

1 Results

We consider finite planar graphs without loops or multiple edges. Let G be a planar graph with vertex set $V(G)$ and edge set $E(G)$. We denote by $|G|$ the order of G, that is, $|G| = |V(G)|$. Given a planar graph G, let P be a set of $|G|$ points in the plane (2-dimentional Euclidean space) in general position (i.e., no three of them are collinear). Then G is said to be *line embedded onto* P or *stright-line embedded onto* P if G can be embedded in the plane so that every vertex of G corresponds to a point of P, every edge corresponds to a straight-line segment, and no two straight-line segments intersect except their common end-point. Namely, G is line embedded onto P if there exists a bijection $\phi : V(G) \to P$ such that two points $\phi(x)$ and $\phi(y)$ are joined by a straight-line segment if and only if x and y are joined by an edge of G and all two distinct open straight-line segments have no point in common. We call such a bijection a *line embedding* or a *straight-line embedding* of G onto P.

In this paper we consider a line embedding having one more property. Let G be a planar graph with n specified viertices $v_1, v_2, \ldots, v_n$, and P a set of $|G|$ points in the plane in general position containing n specified points $p_1, p_2, \ldots, p_n$. Then we say that G is *strongly line embedded onto* P or *strongly straight-line embedded onto* P if G can be line embedded onto P so that for every $1 \leq i \leq n$, v_i corrresponds to p_i, that is, if there exists a line embedding $\phi : V(G) \to P$ such that $\phi(v_i) = p_i$ for all $1 \leq i \leq n$. The line embedding mentioned above is called a *strong line embedding* or a *strong straight-line embedding* of G onto P. A tree with one specified vertex v is usually called a *rooted tree* with root v. Given n disjoint rooted trees T_i with root v_i, $1 \leq i \leq n$, the union $T_1 \cup T_2 \cup \cdots \cup T_n$, whose vertex set is $V(T_1) \cup V(T_2) \cup \cdots \cup V(T_n)$ and whose edge set is $E(T_1) \cup E(T_2) \cup \cdots \cup E(T_n)$, is called a *rooted forest* with roots $v_1, v_2, \ldots, v_n$, which are specified vertices of it.

We begin with the following theorem, which was conjectured by Perles [6] and partially solved by Pach and Törőcsik [7], and whose another simpler proof can be found in Togunaga [8].

Theorem A (Ikebe, Perles, Tamura, and Tokunaga [3]) *A rooted tree T can be*

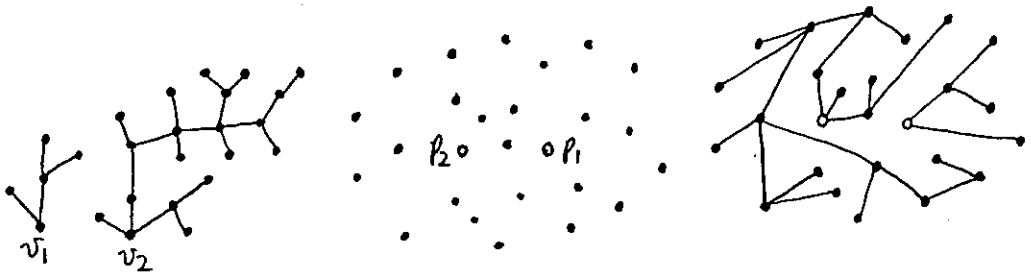


Figure 1: A rooted forest F and its strong line embedding onto P.

Figure 2: A rooted forest F which cannot be strongly line embedded onto P.

strongly line embedded onto every set of $|T|$ points in the plane in general position containing a specified point.

The next theorem gives an algorithm for finding a strong line embedding mentioned in the above theorem.

Theorem B (Bose, McAllister and Snoeyink [1]) *Let T be a rooted tree of order n and P a set of $|T|$ points in the plane in general position containing a specified point. Then we can strongly embed T onto P in $O(n \log n)$ time.*

Our first result is the following.

Theorem 1 (Kaneko and Kano [4]) *A rooted forest F consisting of two rooted trees can be strongly line embedded onto every set of $|F|$ points in the plane in general position containing two specified points (see Figure 1).*

Moreover, our proof of the above theorem gives a $O(|F|^2 \log |F|)$ time algorithm for finding such a strong line embedding. We mention that there exist root forests consisting of four rooted trees which cannot be strongly line embedded onto certain sets of points in the plane in general position containing four specified points. An example of such a forest and a set of points are given in Figure 2. However, we propose the following conjecture.

Conjecture 2 (Kaneko and Kano) *A rooted forest F consisting of three rooted trees can be strongly line embedded onto every set of $|F|$ points in the plane in general position containing three specified points.*

A complete bipartite graph $K(1, k)$ is called a *star*, which is a tree and has one central vertex and k end-vertices. A star with one specified vertex v, which may be an end-vetex, is called a *rooted star* with root v. Our next theorem guarantees the existence of a straight-line embedding of rooted forest consisting of many rooted stars.

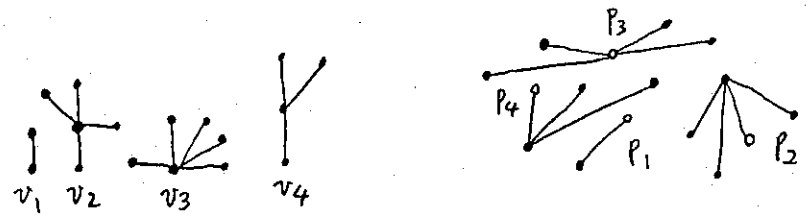


Figure 3: A rooted forest F and its strong line embedding onto P.

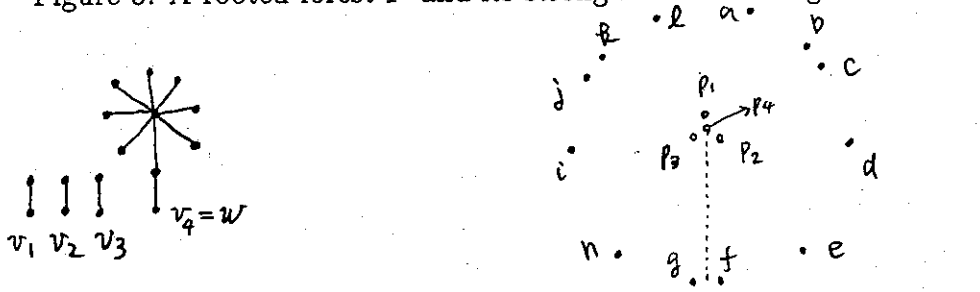


Figure 4: A rooted forest F which cannot be strongly line embedded onto P.

Theorem 3 (Kaneko and Kano [5]) *Let $n \geq 1$ be an integer, and for every $1 \leq i \leq n$, T_i be a rooted star with root v_i. Then a rooted forest $F := T_1 \cup T_2 \cup \ldots \cup T_n$ with roots $v_1, v_2, \ldots, v_n$ can be strongly line embedded onto every set of $|F|$ points in the plane in general position containing n specified points.*

Let us give an example which shows the necessity of the condition that every T_i is a star. For two distinct points x and y in the plane, we denote by $L(x,y)$ the line passing through x and y. Let T_1, T_2, T_3 be rooted stars $K(1,1)$, and let T_4 be a rooted tree which is obtained from the star $K(1,8)$ by adding one new edge together with its new end-vertex u to an end-vertex and whose root is the new end-vertex u (see Figure 4). We define a rooted forest F as $F := T_1 \cup T_2 \cup T_3 \cup T_4$, whose order is 16. Let P be a set of 20 points in the plane in general position containing four specified points p_1, p_2, p_3, p_4 given in Figure 4 that satisfy the following condisiton: (i) p_1, p_2, p_3 are the vertices of a equilateral traingle $\triangle(p_1p_2p_3)$, and p_4 is its center; (i) 16 non-specified points of P lie on the same circle of center p_4; (iii) a line $L(p_1, p_4)$ passes between g and f, (iv) two points a and h lie to the left of $L(p_1, p_3)$; (v) p_2 lies to the left of $L(a,g)$ and of $L(b,h)$; (vi) the above two statements (iv) and (v) symmetricaly hold for all the other points. Then F cannot be strongly line embedded onto P.

Conjecture 4 (Kaneko and Kano) *Let $F := T_1 \cup T_2 \cup \cdots \cup T_n$ a rooted tree with roots $v_1, v_2, \ldots, v_n$ such that $|T_1| \geq |T_2| \geq \cdots \geq |T_n|$, and let P be a set of $|F|$ points in the plane in general position containing n specified points $p_1, p_2, \ldots, p_n$. If for every $1 \leq i \leq n$, p_i is a vertex of conv$(\{p_i, p_{i+1}, \ldots, p_n\})$, then F can be strongly line embedded onto P.*

2 Sketch of Proof of Theorem 1

For a set X of points in the plane, we denote by $\mathrm{conv}(X)$ the convex hull of X, which is the smallest convex set containing X. In order to prove our Theorem 1, we first show the following lemmas.

Lemma 5 *A tree T has a vertex v such that every component of $T - v$ has order less than or equal to $|T|/2$.*

Proof Choose a vetex v of T so that the maximum order of the components of $T - v$ is minimum among all vertices of T. Then v satisfies the condition of this lemma. □

Lemma 6 *Let T be a tree with two specified vertices v_1 and v_2, and P a set of $|T|$ points in the plane in general position containing two specified points p_1 and p_2. If p_1 and p_2 are consecutive vertices of conv(P), then T can be strongly line embedded onto P.*

Proof We prove the lemma by induction on $|T|$. By a suitable rotation of the plane and by the symmetry of p_1 and p_2, we may assume that p_1 lies on the bottom of $\mathrm{conv}(P)$ and that p_2 lies to the left of p_1. Suppose first $\deg_T(v_1) = 1$. In this case, we take a vertex q of $\mathrm{conv}(P \setminus \{p_1\})$ such that a straight-line segment $\overline{qp_2}$ is an edge of $\mathrm{conv}(P \setminus \{p_1\})$ and $\overline{p_1q}$ intersects $\mathrm{conv}(P \setminus \{p_1\})$ at only q. Let u be the vertex of T adjacent to v_1. Then by induction, a tree $T - v_1$ with two specified vertices u and v_2 is strongly line embedded onto $P \setminus \{p_1\}$ with specified points q and p_2. By adding $\overline{p_1q}$ to this embedding, we can get a desired strong line embedding of T.

We next assume $\deg_T(v_1) \geq 2$. Let D be a component of $T - v_1$ not containing v_2. Then $|D| \leq |T| - 2 = |P| - 2$, and so there exists a line l passing through p_1 such that the number of usual points of P lying on or to the right of l is equal to $|D|$. We denote the set of these usaul points of P by Q. Then by Theorem A, a rooted tree $\langle D \cup \{v_1\}\rangle_T$ with root v_1 is strongly line embedded onto $Q \cup \{p_1\}$ with specified point p_1. Furthermore, it follows from the inductive hypothesis that $T - V(D_1)$ with specified vertices v_1 and v_2 is strongly line embedded onto $P \setminus Q$ with specified points p_1 and p_2. By combining the above two embedding, we can obtain a desired strong line embedding of T onto P. □

By the following lemma, when we prove Theorem 1, we may assume that the two roots of $T_1 \cup T_2$ lie inside of $\mathrm{con}(P)$. Namely, the case where one of roots lies on the boundary of $\mathrm{con}(P)$ is easy to prove.

Lemma 7 *Let $T_1 \cup T_2$ be a rooted forest with roots v_1 and v_2, and P a set of $|T_1 \cup T_2|$ points in the plane in general position containing two specified points p_1 and p_2. If p_1 is a vertex of conv(P), then $T_1 \cup T_2$ can be strongly line embedded onto P.*

Proof We prove the lemma by induction on $|T_1 \cup T_2|$. Suppose first $\deg_{T_1}(v_1) = 1$. We take a vertex q of $\mathrm{conv}(P \setminus \{p_1\})$ such that $q \neq p_2$ and $\overline{p_1q}$ intersects $\mathrm{conv}(P \setminus \{p_1\})$ at only q. Let u be the vertex of T_1 adjacent to v_1. Then by induction, a rooted forest $(T_1 - v_1) \cup T_2$ with roots u and v_2 can be strongly line embedded onto $P \setminus \{p_1\}$ with

specified points q and p_2. By adding $\overline{p_1q}$ to this embedding, we get a desired embedding of $T_1 \cup T_2$ onto P.

We next assume $\deg_{T_1}(v_1) \geq 2$. Let D be one of the smallest components of $T_1 - v_1$. Then $|D| \leq (|T_1| - 1)/2 \leq (|P| - 2)/2$, and so at least one of regions detemined by a line passing through p_1 and p_2 contains at lest $|D|$ usual points of P. Thus there exists a line l passing through p_1 such that an open region R determined by l contains exactly $|D|$ usual points of P and does not contain p_2. Then by Theorem A, $\langle D \cup \{p_1\}\rangle_{T_1}$ with root v_1 is strongly line embedded onto $(R \cap U(P)) \cup \{p_1\}$ with specified point p_1. By the inductive hypothesis, $(T_1 - V(D_1)) \cup T_2$ with roots v_1 and v_2 is strongly line embedded onto $P \setminus (R \cap U(P))$ with specified points p_1 and p_2. By these embeddings, we can obtain a desired strong line embedding of $T_1 \cup T_2$ onto P. □

Lemma 8 *Let $F := T_1 \cup T_2 \cup T_3$ be a rooted forest with roots v_1, v_2, v_3, and P a set of $|F|$ points in the plane in general position containing three specified points p_1, p_2, p_3. If p_1 and p_2 are consecutive vertices of conv(P), then F can be strongly line embedded onto P.*

Proof Without loss of generality, we may assume that p_1 lies on the bottom of conv(P) and that p_2 lies to the left of p_1. We consider only rays r emanating from p_1 and going upward, and so a ray means such a ray. Given a ray r, let $P(r)$ denote the set of points of P lying on or to the right of r. Then $p_1 \in P(r)$ for every ray r, and there exists a ray r_1 such that either (i) $|P(r_1)| = |T_1|$ and $P(r_1)$ does not contain p_3; or (ii) $|P(r_1)| = |T_1| + |T_3|$ and $P(r_1)$ contains p_3. If r_1 satisfies (i), then T_1 and $T_2 \cup T_3$ are strongly line embedded onto $P(r_1)$ and onto $P \setminus P(r_1)$, respectively, by Theorem A and by Lemma 7. Similarly, if r_1 satisfies (ii), then $T_1 \cup T_3$ and T_2 are strongly line embedded onto $P(r_1)$ and onto $P \setminus P(r_1)$, respectively. Therefore $T_1 \cup T_2 \cup T_3$ can be strongly line embedded onto P. □

Sketch of Proof of Theorem 1 When we prove the theorem, we prove the following calims and some other calimes, and consider two cases.

Claim 1. We may assume that for every line l passing through p_1, the number $f(l)$ of usual points of P lying on or to the left of l is less than n_1.

Define an integer M by

$$M := \max\{f(l)\},$$

where maximum is taken over all the lines l passing through p_1 except the horizontal line passing through p_1 and p_2, and $f(l)$ is defined as in Claim 1. Then $(n_1 + n_2)/2 \leq M < n_1$ by Claim 1. Let $D_1, D_2, \ldots, D_r$ be the components of $T_1 - v_1$ such that $|D_1| \geq |D_2| \geq \cdots \geq |D_m|$.

Claim 2. We may assume $|D_1| > M$.

By Lemma 5, D_1 has a vetex w_1 such that each component of $D_1 - w_1$ has order less than or equal to $|D_1|/2$. Let $A_1, A_2, \ldots, A_t$ be the components of $D_1 - w_1$ such that A_1 is the component containing the vertex adjacent to v_1 in T_1 and A_2 is a largest component among $A_2, A_3, \ldots, A_t$. Note that $A_1 = \emptyset$ if v_1 and w_1 are adjacent in T_1. Let $w_2 \in A_2$ be the vertex adjacent to w_1. Define $B_4 := A_2$ and

$$B_2 := A_1 \cup (\bigcup_{i=3}^{r} A_i) \quad \text{if } t \geq 3, \qquad \text{and otherwise } B_2 := A_1,$$

where an integer r, $3 \leq r \leq t$, is chosen as large as possible subject to $|B_2| < M$, in particular, if $r < t$, then $|B_2 \cup A_{r+1}| \geq M$. Note that $|A_1| < M$ as $|A_1| \leq |D_1|/2 \leq (n_1 - 1)/2 \leq (n_1 + n_2)/2 - 1 \leq M - 1$. We define

$$B_3 := A_{r+1} \cup \cdots \cup A_t \quad \text{if } r < t, \qquad \text{and otherwise } B_3 := \emptyset.$$

Moreover, put

$$B_1 := D_2 \cup \cdots \cup D_m = T_1 - (V(D_1) \cup \{v_1\}).$$

Then $V(T_1) = V(B_1) \cup V(D_1) \cup \{v_1\}$, $V(D_1) = V(B_2) \cup V(B_3) \cup V(B_4) \cup \{w_1\}$ and $w_2 \in B_4$.

Claim 3. We may assume that for every line l passing through p_1, the number $f(l)$ of usual points of P lying on or to the left of l is greater than or equal to $|B_2| + 2$.

Claim 4. We may assume $B_3 \neq \emptyset$.

References

[1] P. Bose, M. McAllister and J. Snoeyink, Optimal Algorithms to embed trees in a point set, *Graph drawing* (Proceedings of Symposium on Graph Drawding, GD'95), **1027** Lecture Notes in Computer Science.

[2] H. de Fraysseix, P. Pach and R. Pollack, How to draw a planar graph on a grid, *Combinatorica*, **10** (1990) 41-51.

[3] Y. Ikeba, M. Perles, A. Tamura and S. Tokunaga, *The rooted tree embedding problem into points on the plane*, Discrte Comput. Geom. **11** (1994) 51-63.

[4] A. Kaneko and M. Kano, A straight-line embedding of two rooted trees in the plane, In preparation.

[5] A. Kaneko and M. Kano, Straight line embeddings of star forests in the plane, In preparation.

[6] M. Perles, Open problem proposed at the DIMACS Workshop on Arrangements, Rutgers University, 1990.

[7] J. Pach and J. Törőcsik, Layout of rooted tree, *Planar Graphs* (DIMACS Series in Discrete Math. and Theoritical Comput. Sci.) **9** (1993) 131- 137.

[8] S.Tokunaga, On a straight-line embedding problem of graphs, *Discrete Math.* **150** (1996) 371-378.

A balanced partition of points in the plane and tree embedding problems (Extended abstract)

Atsushi Kaneko
Department of Electronic Engineering
Kogakuin University
Nishi-Shinjuku, Shinjuku-ku, Tokyo 163-91 Japan
e-mail: kaneko@ee.kogakuin.ac.jp

1 Introduction

Let $G = (V,\ E)$ be a graph and let P be a set of points in the plane in general position (no three points collinear). We denote by $|G|$ the order of G and by $\mathrm{conv}(P)$ the convex hull of P. We say that G can be *straight-line embedded onto* P, if there exists a one-to-one mapping $\phi : V \to P$ such that two points $\phi(x)$ and $\phi(y)$ are joined by a straight line segment if and only if $xy \in E$ and such that any two distinct open straight line segments have no point in common. Furthermore, let G be a planar graph with n specified vertices $v_1, v_2, \cdots, v_n$, and P a set of $|G|$ points in the plane in general position containing n specified points $p_1, p_2, \cdots, p_n$. Then we say that G can be *strongly line embedded onto* P if G can be straight-line embedded onto P so that v_i corresponds to p_i, $1 \leq i \leq n$. Let $T_1, T_2, \cdots, T_n$ be n disjoint rooted trees such that v_i is the root of T_i, $1 \leq i \leq n$. Then the union $T_1 \cup T_2 \cup \cdots \cup T_n$ is called a *rooted forest with roots* $v_1, v_2, \cdots, v_n$. ($\{v_1, v_2, \cdots, v_n\}$ are specified vertices of the forest.)

Ikebe et al. solved the rooted tree embedding problem, which was originally posed by Perles at the 1990 DIMACS workshop on arrangements. Short proofs of this theorem have been given in [1] and [4].

Theorem A (Ikebe, Perles, Tamura,and Tokunaga [3]) *A rooted tree can be strongly line embedded onto any set of $|T|$ points in the plane in general position containing a specified point.*

2 Thorems

In this paper, we prove the following theorems.

Theorem 1 (Kaneko and Kano) *Let $T_1, T_2, \cdots, T_m$ be m disjoint rooted trees such that $||T_i| - |T_j|| \leq 1$ for any $1 \leq i < j \leq m$ and let v_i be the root of T_i, $1 \leq i \leq m$. Let R and S be two disjoint sets of points in the plane such that $|R \cup S| = \sum_{k=1}^{m} |T_k|$ and $S = \{p_1, p_2, \cdots, p_m\}$. Then the rooted forest $T_1 \cup T_2 \cup \cdots \cup T_m$ with roots $v_1, v_2, \cdots, v_m$ can be strongly line embedded onto $R \cup S$ with S being specified points.*

In order to prove Theorem 1 we need the next theorem, which is of some general interest.

Theorem 2 (Kaneko and Kano) *Let S_1, S_2 and R be three disjoint sets of points in the plane such that no three points of $S_1 \cup S_2 \cup R$ are collinear and $|S_1 \cup S_2| = q$. Let m be a positive integer. If $|T| = (m-1)|S_1| + m|S_2|$, then $S_1 \cup S_2 \cup R$ is partitioned into q sets $P_1, P_2, \cdots, P_q$ such that*

(1) $|P_i \cap (S_1 \cup S_2)| = 1$, for all $1 \leq i \leq q$,
(2) $\mathrm{conv}(P_i) \cap \mathrm{conv}(P_j) = \phi$ for all $1 \leq i < j \leq q$, and
(3) $|P_i| = m$ if $|P_i \cap S_1| = 1$, and $|P_i| = m+1$ if $|P_i \cap S_2| = 1$.

Combining Theorem 2 with Theorem A, we deduce Theorem 1. It seems that we can easily prove Theorem 2 by using the ham-sandwich theorem (see, for example [2]). However, it may happen that q is odd and the non-trivial part of the proof of Theorem 2 is to deal with this case.

3 Conjectures

In view of Theorem 2 we give the following conjectures.

Conjecture B (Kaneko and Kano) *Let S and R be two disjoint sets of points in the plane such that no three points of $S \cup R$ are collinear and $|S| = 2q$. Let $m \geq 2$ be an integer. If $|T| = mq$, then $S \cup R$ is partitioned into q sets $P_1, P_2, \cdots, P_q$ such that*

(1) $|P_i \cap S| = 2$ for all $1 \leq i \leq q$,
(2) $\mathrm{conv}(P_i) \cap \mathrm{conv}(P_j) = \phi$ for all $1 \leq i < j \leq q$, and
(3) $|P_i| = m+2$ for all $1 \leq i \leq q$.

Conjecture C (Kaneko and Kano) *Let m and n be two integers such that $m \geq n \geq 2$. Let S and R be two disjoint sets of points in the plane such that no three points of $S \cup R$ are collinear and $|S| = nq$. If $|T| = mq$, then $S \cup R$ is partitioned into q sets $P_1, P_2, \cdots, P_q$ such that*

(1) $|P_i \cap S| = n$ for all $1 \leq i \leq q$,
(2) $\mathrm{conv}(P_i) \cap \mathrm{conv}(P_j) = \phi$ for all $1 \leq i < j \leq q$, and
(3) $|P_i| = m+n$ for all $1 \leq i \leq q$.

Conjecture D (Kaneko and Kano) *Let S_1, S_2, S_3 and R be four disjoint sets of points in the plane such that no three points of $S_1 \cup S_2 \cup S_3 \cup R$ are collinear and $|S_1 \cup S_2 \cup S_3| = q$. Let m be a positive integer. If $|T| = (m-1)|S_1| + m|S_2| + (m+1)|S_3|$, then $S_1 \cup S_2 \cup S_3 \cup R$ is partitioned into q sets $P_1, P_2, \cdots, P_q$ such that*

(1) $|P_i \cap (S_1 \cup S_2 \cup S_3)| = 1$, *for all* $1 \leq i \leq q$,

(2) $\mathrm{conv}(P_i) \cap \mathrm{conv}(P_j) = \phi$ *for all* $1 \leq i < j \leq q$, *and*

(3) $|P_i| = m$ *if* $|P_i \cap S_1| = 1$,
$|P_i| = m+1$ *if* $|P_i \cap S_2| = 1$, *and*
$|P_i| = m+2$ *if* $|P_i \cap S_3| = 1$.

If conjecture D is true, it is easy to see that we get an extension of Theorem 1 by the same argument.

References

[1] P. Bose, M. McAllister and J. Snoeyink, Optimal algorithms to embed trees in a point set, *Proc. Graph Drawing (GD'95), Lecture Notes in Comp. Science* 1190, Springer Verlag (1995), pp. 64-75.

[2] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer Verlag, Berlin, 1987.

[3] Y. Ikebe, M. Perles, A. Tamura and S. Tokunaga, The rooted tree embedding problem into points on the plane, *Discrete Comput. Geom.* **11** (1994), pp. 51-63.

[4] S. Tokunaga, On a straight-line embedding problem of graphs, *Discrete Mathematics* **150** (1996) pp. 371-378.

Parallel Algorithms for Longest Increasing Chains in the Plane and Related Problems

Mikhail J. Atallah* Danny Z. Chen† Kevin S. Klenk‡

Abstract

Given a set S of n points in the plane such that each point in S is associated with a nonnegative weight, we consider the problem of computing the single-source longest increasing chains among the points in S. This problem is a generalization of the planar maximal layers problem. In this paper, we present a parallel algorithm that computes the single-source longest increasing chains in the plane in $O(\log^2 n)$ time using $O(n^2/\log^3 n)$ processors in the CREW PRAM computational model. We also solve a related problem of computing the all-pairs longest paths in an n-node weighted planar st-graph, in $O(\log^2 n)$ time using $O(n^2/\log n)$ CREW PRAM processors. Both of our parallel algorithms are improvement over the previously best known results.

1 Introduction

A point p in the plane is said to *dominate* another point q if and only if $x(p) \geq x(q)$ and $y(p) \geq y(q)$, where $x(p')$ and $y(p')$ respectively denote the x- and y-coordinates of a point p'. Let S be a set of n points in the plane such that each point p in S is associated with a nonnegative weight $w(p)$, and let $\sigma = (p_1, p_2, \ldots, p_k)$ be a sequence of distinct points in S. The sequence σ is *increasing* if and only if p_i dominates p_{i-1} for all i, $1 < i \leq k$. We also call σ an increasing chain. The *length* of σ is the sum of the weights of the points in σ. The chain σ between p_1 and p_k is *longest* if no other p_1-to-p_k increasing chain passing through the points in S has a length greater than σ. In this paper, we study the problem of computing in parallel longest increasing chains in S and some related problems.

The notions of dominance between points and of increasing chains are useful in many problems in computational geometry, graph theory, scheduling, and economics, including problems on independent dominating sets in permutation graphs and problems on increasing subsequences of a sequence of numbers (cf. [5] for more on these). The problem of computing a longest increasing subsequence of a number sequence, for instance, has appeared in a number of areas, and there are $O(n \log n)$ time sequential algorithms for this problem (see [10–12] among others). In particular, increasing subsequence problems occur in the context of circle graphs, circular-arc graphs, interval graphs, and permutation graphs (see [3] for references). In this paper, we shall also exploit a connection between increasing chains and paths in st-graphs.

For sake of simplicity, we shall henceforth refer to all increasing chains simply as *chains*.

Some interesting work has been done on solving various chain problems. Atallah and Kosaraju [5] presented optimal sequential $O(n \log n)$ time algorithms for several problems related to planar chains. Atallah and Chen [3] gave a sequential $O(n^2)$ time algorithm for the unweighted planar all-pairs longest chains and a parallel algorithm for the weighted planar all-pairs version in $O(\log^2 n)$ time using $O(n^2/\log n)$ CREW PRAM processors.

The *single-source* longest chains problem is that

*Dept. of Computer Sciences, Purdue University, West Lafayette, IN 47907, USA, E-mail: mja@cs.purdue.edu. This author gratefully acknowledges the support of the COAST Project at Purdue University and its sponsors, in particular Hewlett Packard, DARPA, and the National Security Agency.

†Dept. of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556-5637, USA, E-mail: chen@cse.nd.edu. This research of this author was supported in part by the National Science Foundation under Grant CCR-9623585.

‡Dept. of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556-5637, USA, E-mail: kklenk@cse.nd.edu. This research of this author was supported in part by the National Science Foundation under Grant CCR-9623585.

of finding longest chains from a fixed *source* point of S to all other points of S. The single-source problem is in fact a generalization of the maximal layers problem, which computes the *maximal layers* of the points in S as follows: find all the points of S that are not dominated by any other point of S, call these *layer-1 points*, and remove them from S; produce points of subsequent layers by repeating this process on S, until S becomes empty. It is easy to reduce the maximal layers problem to the unweighted single-source longest chains problem (i.e., by letting the weight of each point be one unit). Optimal sequential $O(n \log n)$ time solutions for the planar maximal layers problem easily follow from the known algorithms for the longest increasing subsequence [10–12]; an $O(n \log n)$ time algorithm for the 3-D version of the problem was recently given by Atallah, Goodrich, and Ramaiyer [4]. Aggarwal and Park [1] gave a parallel algorithm for the planar maximal layers problem that takes $O(\log^2 n)$ time and $O(n^2/\log n)$ CREW PRAM processors.

This paper extends Atallah and Chen's parallel approach [3] to computing the planar single-source longest chains. One of the ways in which this extension differs from [3] is that it involves a new *implicit spreading* idea (to be described in Section 2) that helps us overcome the difficulty of extending the solution for a smaller size problem to one for a larger size problem (this need did not occur in [3]). Another idea involves a different kind of partitioning scheme, leading to a logarithmic number of iterations that, while performed in sequence, individually involve parallelism in the way they are performed. Our parallel single-source algorithm takes $O(\log^2 n)$ time using $O(n^2/\log^3 n)$ processors. Our solution, when applied to the (simpler) planar maximal layers problem, improves the processor bound of Aggarwal and Park [1] by a $\log^2 n$ factor while retaining the same time bound.

We also show a new application of the parallel all-pairs longest chains algorithm by Atallah and Chen [3], to computing the all-pairs *longest paths* in weighted planar st-graphs. Briefly, a planar st-graph is a planar directed acyclic graph with exactly one source s and exactly one sink t that is embedded in the plane such that s and t are both on the boundary of the outer face [18, 19]. Planar st-graphs have been used in many applications such as: computational geometry, graph drawing, motion planning, partial orders, planar graph embedding, and VLSI layout (see [19] for references).

We are not aware of any previous parallel algorithm for computing the all-pairs longest paths in weighted planar st-graphs. But, one could attempt to apply known parallel algorithms for *shortest paths* in directed graphs to solve this st-graph problem. Han, Pan, and Reif [13] presented an algorithm that computes all-pairs shortest paths in general directed graphs in $O(\log^2 n)$ time using $o(n^3/\log^2 n)$ EREW PRAM processors. Cohen [8] gave algorithms that compute shortest paths in planar directed graphs, in $O(\log^4 n)$ time and $O(n^2/\log^3 n)$ CREW PRAM processors.

Our algorithm takes advantage of the underlying geometry of a planar st-graph to enable us to compute the all-pairs longest paths in an n-node weighted planar st-graph in $O(\log^2 n)$ time using $O(n^2/\log n)$ CREW PRAM processors. Specifically, our computation is performed by reducing the all-pairs longest paths in a planar st-graph to that of the all-pairs longest chains in the plane. Due to the space limit, we leave the st-graph algorithm to the full paper.

In the rest of this paper, we will focus on computing the *lengths* of the longest chains/paths. Our algorithms can be easily modified to generate the actual longest chains/paths as well as their lengths, by using the techniques for finding actual paths in [3].

2 Preliminaries

As in [3], our solution is based on fast matrix multiplications of particular types of matrices (specifically, *monotone* matrices) in the $(\max, +)$ closed semi-ring, i.e., $(M' \times M'')(i,j) = \max_k\{M'(i,k) + M''(k,j)\}$. All of our matrix multiplications are of this form.

Atallah and Chen [3] considered the problem of computing an $n \times n$ matrix D of the lengths of the longest chains between each pair of points in S. Thus, $D(p,q)$ gives the length of a longest p-to-q chain, for any $p, q \in S$. The computation of D is based on the following observation.

Lemma 1 (**[3]**) *Let V_l, V_m and V_r be three vertical lines with $x(V_l) < x(V_m) < x(V_r)$. Let S_l (resp., S_r) be the set of points in S whose x-coordinates are $\geq x(V_l)$ (resp., $\geq x(V_m)$) and $\leq x(V_m)$ (resp., $\leq x(V_r)$). Let the set X_l (resp., X_r) contain the horizontal projections of the points of S_l (resp., S_r) onto V_l (resp., V_r), and X_m contain the horizontal projections of the points of $S_l \cup S_r$ onto*

V_m. *Let the weights of the points in* X_l *(resp.,* X_m, X_r*) be all zero. Let* $\Omega = S_l \cup S_r \cup X_l \cup X_m \cup X_r$. *Then for every increasing chain* C *through the points in* Ω *from a point* $p \in X_l$ *to a point* $q \in X_r$, $y(p) \leq y(q)$, *there is a* p*-to-*q *increasing chain* C' *through* Ω *such that* C' *is at least as long as* C *and* C' *goes through some point* $w \in X_m$.

Let $M[A, B]$ denote a matrix that contains the lengths of longest chains starting from a point in the set A, ending at a point in the set B, and passing through a particular set of points. Lemma 1 implies that $M[X_l, X_r] = M[X_l, X_m] \times M[X_m, X_r]$. Atallah and Chen [3] also showed that the matrices $M[X_l, X_r]$, $M[X_l, X_m]$, and $M[X_m, X_r]$ have special properties that enable fast matrix multiplications (by using the monotone matrix searching algorithms in [1, 2]). This is summarized in the next lemma.

Lemma 2 ([3]) *Let* V_l, V_m, V_r, X_l, X_m, X_r, *and* Ω *be defined as in Lemma 1. Let the point set* X_l *(resp.,* X_m, X_r*) be ordered by increasing* y*-coordinates along* V_l *(resp.,* V_m, V_r*). Assume that the size* $|X_l|$ *of* X_l *is proportional to* $|X_r|$. *Then, given the matrices* $M[X_l, X_m]$ *and* $M[X_m, X_r]$, *the matrix* $M[X_l, X_r]$ *can be computed in* $O(\log |X_l|)$ *time and* $O(|X_l|^2 / \log |X_l|)$ *CREW PRAM processors.*

Lemma 2 was the basis for a two-phase algorithm for computing the all-pairs planar longest chains in [3]. Our algorithm uses one phase of this algorithm. Hence, we sketch the necessary portions of this two-phase algorithm.

Let $S = \{p_1, p_2, \ldots, p_n\}$. Without loss of generality (WLOG), we assume that $x(p_1) < x(p_2) < \cdots < x(p_n)$. Let $V_0, V_1, \ldots, V_n$ be vertical lines such that $x(V_0) < x(p_1), x(p_n) < x(V_n)$, and $x(p_i) < x(V_i) < x(p_{i+1})$ for all $i \in \{1, 2, \ldots, n-1\}$. Let T be a complete n-leaf binary tree. For the i-th leaf v of T in the left-to-right order, associate with v the region I_v of the plane that is between V_{i-1} and V_i. For each internal node v of T, associate with v the region I_v consisting of the union of the regions of its children. That is, if v has children u and w, then $I_v = I_u \cup I_w$. The tree T is called the *computation tree* on the point set S.

Let v be a node of T. Suppose that the left (resp., right) boundary of I_v is V_i (resp., V_j), and let $S_v = S \cap I_v$, that is, S_v is the subset of the points of S that lie in the region I_v. Let L_v (resp., R_v) be the set consisting of the horizontal projections of S_v onto V_i (resp., V_j). If v is an internal node of T with left (resp., right) child u (resp., w), then let Y_v denote $R_u \cup L_w$, i.e., the horizontal projections of the points of S_v onto the vertical line that separates the region I_u from the region I_w. The weights of all projection points are zero.

Atallah and Chen's algorithm proceeds in the following two phases. In Phase 1, start at the leaves and go up the tree T level by level, computing, at each level, the $M[L_v, R_v]$ matrices for nodes v of T, which contain the lengths of all the L_v-to-R_v longest chains (these chains begin on L_v and end on R_v, possibly going through points in S_v along the way). For an internal node v of T, $M[L_v, R_v]$ is computed from the two matrices $M[L_v, Y_v]$ and $M[Y_v, R_v]$ based on Lemma 2. Note that when the computation reaches v, only the matrix $M[L_u, R_u]$ (resp., $M[L_w, R_w]$) is available from its left child u (resp., right child w). The matrices $M[L_v, Y_v]$ and $M[Y_v, R_v]$ need to be obtained from $M[L_u, R_u]$ and $M[L_w, R_w]$, respectively. $M[L_v, Y_v]$ is computed, in $O(\log n)$ time and $O(|S_v|^2 / \log n)$ processors, from $M[L_u, R_u]$ by the following *spreading procedure* (the computation of $M[Y_v, R_v]$ is similar):

1. Perform a parallel prefix along L_v (resp., Y_v), in decreasing (resp., increasing) y-coordinates, to compute, for every point $z \in L_v$ (resp., Y_v), the lowest (resp., highest) point $l(z)$ (resp., $h(z)$) such that $l(z) \in L_u$ (resp., $h(z) \in R_u$) and that $y(l(z)) \geq y(z)$ (resp., $y(h(z)) \leq y(z)$).

2. For every pair of points p and q such that $p \in L_v$ and $q \in Y_v$, do the following: If $y(p) \leq y(l(p)) \leq y(q)$, then let $M[L_v, Y_v](p, q) = M[L_u, R_u](l(p), h(q))$. Otherwise, let $M[L_v, Y_v](p, q) = 0$.

The matrices $M[L_v, Y_v]$, $M[Y_v, R_v]$, and $M[L_v, R_v]$ are stored at v (even after the computation has reached higher level nodes of T); these matrices are useful in Phase 2. Phase 1 is accomplished in $O(\log^2 n)$ time, $O(n^2 / \log^2 n)$ processors, and $O(n^2)$ space.

Phase 2 is a top-down computation, starting at the root of T and going downward to the leaves, one level at a time. This phase uses the information produced in Phase 1 to obtain the lengths of the all-pairs longest chains. In particular, for every pair of nodes u, w at the same level of T such that u is to the left of w, it computes the matrix $M[R_u, L_w]$. This phase takes $O(\log^2 n)$ time, $O(n^2 / \log n)$ processors, and $O(n^2)$ space.

Observe that, although the above spreading procedure *explicitly* obtains the matrix $M[L_v, Y_v]$ from the matrix $M[L_u, R_u]$ in [3], it is possible to represent $M[L_v, Y_v]$ *implicitly*. That is, the matrix $M[L_v, Y_v]$ can be fully described by $M[L_u, R_u]$ and the two sorted lists L_v and Y_v. To obtain this representation of $M[L_v, Y_v]$, we only need to perform Step 1 of the spreading procedure (but not Step 2). This takes $O(\log(|L_v| + |Y_v|))$ time and $O((|L_v| + |Y_v|)/\log(|L_v| + |Y_v|))$ processors. After that, useful information about the matrix $M[L_v, Y_v]$ is readily available. For example, every entry $M[L_v, Y_v](p, q)$ can be computed in $O(1)$ time and one processor from $M[L_u, R_u]$, as in Step 2 of the above spreading procedure. We call this the *implicit spreading procedure*. Although it would not make much difference in the overall algorithm of [3], the implicit spreading procedure is important to our single-source algorithm in the next section. Specifically, the implicit spreading procedure is useful when $|L_u| + |R_u| = o(|L_v| + |Y_v|)$ (i.e., when $|L_u|+|R_u|$ is asymptotically smaller than $|L_v|+|Y_v|$).

3 Single-source longest chain algorithm

This section presents the $O(\log^2 n)$ time, $O(n^2/\log^3 n)$ processor algorithm for computing the single-source longest chains in the plane. WLOG, we assume that the source point p^* dominates all other points in S. We also assume that we have already sorted the points in S by their x-coordinates and by their y-coordinates, in $O(\log n)$ time and $O(n)$ processors [9].

The all-pairs longest chain algorithm in [3] relies heavily on the multiplication of two $m \times m$ matrices via monotone matrix searching. For our single-source computation, we often rely on multiplying an $m \times m$ matrix with an $m \times 1$ matrix. The following lemma is for this purpose:

Lemma 3 *Let V_l and V_r be two vertical lines such that $x(V_l) < x(V_r)$. Let S' be the set of points of S that are between V_l and V_r, with $m = |S'|$. Let X_l (resp., X_r) be the set of horizontal projection points of S' onto V_l (resp., V_r) that are ordered by increasing y-coordinates and whose weights are all zero. Then, given the $m \times m$ matrix $M[X_l, X_r]$ and the $m \times 1$ matrix $M[X_r, p^*]$ (for lengths of the longest chains through the points of S), the $m \times 1$ matrix $M[X_l, p^*]$ can be computed in $O(\log m)$ time and $O(m)$ EREW PRAM processors.*

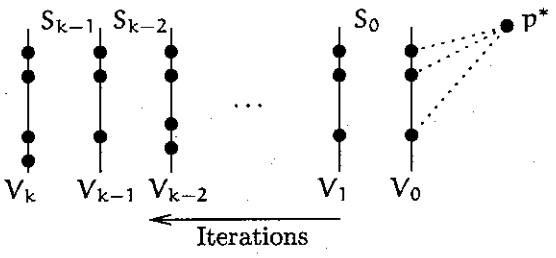


Figure 1: Illustrating the single-source longest chain algorithm.

Proof: This is a straightforward adaptation of Lemma 2 to the single-source situation (one could consider the source point p^* being also on a vertical line). In this case, multiplying an $m \times m$ length matrix by an $m \times 1$ matrix is the main operation, which can be done by using Atallah and Kosaraju's $O(\log m)$ time, $O(m)$ processor algorithm for searching an $m \times m$ monotone matrix [6]. □

Our algorithm for computing the single-source longest chains in the plane works in an iterative fashion, taking advantage of the monotonicity shown in Lemmas 1 and 3. The main steps of the algorithm are as follows:

1. Partition the set $S - \{p^*\}$ of $n - 1$ points into k subsets of (roughly) size $n/\log n$ each, by using $k = \log n$ vertical lines $V_0, V_1, \ldots, V_k$, in the *right-to-left* order (see Figure 1). WLOG, we assume that no point of S is on any line V_i and that the source point p^* is the only point of S to the right of the line V_0.

2. Let S_i be the subset of the points in S that lie between the vertical lines V_i and V_{i+1}. Project horizontally the points of S_i onto V_i and V_{i+1} (we call these projection points the *boundary points* for S_i). Let the weights of all the projection points be zero.

3. Compute the lengths of the longest chains from the projection points on V_i to the projection points on V_{i+1} (through the points of S_i).

4. For $i = 0, 1, \ldots, k-1$, iteratively compute the lengths of the longest chains from the source point p^* to the boundary points on each vertical line V_i.

5. Compute the lengths of the longest chains from p^* to the points in every subset S_i.

We now discuss the details of the steps of the above algorithm. Steps 1 and 2 can be done easily in $O(\log n)$ time using $O(n)$ processors. Hence we only need to focus on Steps 3, 4, and 5.

The computation of the longest chain lengths in Step 3 from the projection points on V_i to the projection points on V_{i+1} can be performed in parallel for every $i = 0, 1, \ldots, k-1$, by using Phase 1 of the algorithm in [3]. We run Phase 1 of [3] on each point set S_i, generating and maintaining a $|S_i|$-leaf computation tree T_i on S_i, together with all the length matrices produced during this process. Let $R(S_i)$ (resp., $L(S_i)$) be the set of projection points of S_i onto V_i (resp., V_{i+1}). Then after Step 3, the length matrix $M[L(S_i), R(S_i)]$ is available at the root node of the tree T_i. Each tree T_i and the length matrices stored at its nodes are useful in the subsequent steps of the algorithm. Phase 1 of the algorithm in [3] takes $O(\log^2 m)$ time, $O(m^2/\log^2 m)$ processors, and $O(m^2)$ space on a set of m points. Thus, the computation on each point set S_i (whose size is $O(n/\log n)$) in Step 3 takes $O(\log^2 n)$ time, $O(n^2/\log^4 n)$ processors, and $O(n^2/\log^2 n)$ space. Summing over all the $k = \log n$ sets S_i, this step takes $O(\log^2 n)$ time, $O(n^2/\log^3 n)$ processors, and $O(n^2/\log n)$ space.

Step 4 computes the longest chain lengths from the boundary points of each point set S_i to the source point p^*. This computation is done *iteratively*, as follows. Let P_i be the set of the horizontal projection points of all the points in S onto the vertical line V_i (thus $|P_i| = n$). Then for every $i = 0, 1, \ldots, k-1$, compute, by using the implicit spreading procedure in Section 2, the (implicitly represented) length matrix $M[P_{i+1}, P_i]$ of size $n \times n$ from the $(n/\log n) \times (n/\log n)$ matrix $M[L(S_i), R(S_i)]$ (from Step 3, $M[L(S_i), R(S_i)]$ is already available). This takes $O(\log n)$ time and $O(n/\log n)$ processors for each i, and $O(\log n)$ time and $O(n)$ processors for all the $k = \log n$ instances. (Note that, if we had used an *explicit* representation for every $n \times n$ matrix $M[P_{i+1}, P_i]$, then it would have taken $O(\log n)$ time and $O(n^2/\log n)$ processors to obtain each such matrix, and $O(\log n)$ time and $O(n^2)$ processors to obtain all the $\log n$ such matrices, clearly too expensive an approach!) Next, the following iterative procedure is performed:

> First, let the length matrix $M[P_0, p^*]$ be such that every entry $M[P_0, p^*](p, p^*) = w(p^*)$, where $w(p^*)$ is the weight of p^*. This is because there is no point of $S - \{p^*\}$ to the right of the line V_0 (see Figure 1). For any $i \in \{0, 1, \ldots, k-2\}$, once the matrix $M[P_i, p^*]$ is available, compute $M[P_{i+1}, p^*]$ by multiplying the $n \times n$ matrix $M[P_{i+1}, P_i]$ with the $n \times 1$ matrix $M[P_i, p^*]$, in $O(\log n)$ time and $O(n)$ processors based on Lemma 3.

Finally, extract the matrix $M[R(S_i), p^*]$ from $M[P_i, p^*]$, for every $i \in \{0, 1, \ldots, k-1\}$. Since there are $k - 1 = O(\log n)$ iterations to perform, Step 4 takes altogether $O(\log^2 n)$ time and $O(n)$ processors.

For Step 5, recall that for every $i \in \{0, 1, \ldots, k-1\}$, we have maintained (in Step 3) the computation tree T_i on the point set S_i together with a collection of length matrices stored at the nodes of T_i. These length matrices were computed on S_i by using Phase 1 of the algorithm in [3]. Step 5 uses these matrices to compute the longest chain lengths from p^* to all the points in every S_i.

In Step 5, a top-down computation is performed on each tree T_i. For an internal node v of T_i, let v be associated with the region I_v of the plane that is between the vertical lines $H_l(v)$ and $H_r(v)$. For example, the root of T_i is associated with the region bounded by the vertical lines V_i and V_{i+1}. Let L_v (resp., R_v) be the set consisting of the horizontal projections of $S \cap I_v$ onto $H_l(v)$ (resp., $H_r(v)$). Let $H_m(v)$ be the vertical line separating the regions I_u and I_w, where u (resp., w) is the left (resp., right) child of v, and let Y_v be the set consisting of the horizontal projections of $S \cap I_v$ onto $H_m(v)$. Suppose that the top-down computation now reaches the node v and assume that the length matrix $M[R_v, p^*]$ is already available (note that, initially, the matrix $M[R(S_i), p^*]$ is available from Step 4). At v, the matrix $M[Y_v, p^*]$ is first computed; this is done by multiplying the matrix $M[Y_v, R_v]$ (available from Step 3) with the matrix $M[R_v, p^*]$, in $O(\log |R_v|)$ time and $O(|R_v|)$ processors based on Lemma 3. The matrix $M[R_u, p^*]$ is then extracted from $M[Y_v, p^*]$ and the matrix $M[R_w, p^*]$ is extracted from $M[R_v, p^*]$. After that, the computation is carried out recursively at each child of v. Step 5 takes $O(\log^2 n)$ time and $O(n)$ processors.

Summing over all the five steps, our parallel single-source algorithm presented above takes $O(\log^2 n)$ time, $O(n^2/\log^3 n)$ CREW PRAM processors, and $O(n^2/\log n)$ space. As we mentioned before, this algorithm also solves the maximal layers problem in the same complexity bounds (by letting the weights of all the points of S be a

unit). In comparison, Aggarwal and Park's CREW PRAM algorithm for the maximal layers problem [1] takes $O(\log^2 n)$ time, $O(n^2/\log n)$ processors, and $O(n^2)$ space.

Remark: We should point out that there is a trade-off between the time and processor/space bounds in our parallel single-source algorithm. The trade-off is determined by the value of the parameter k. By using a larger value of k, one can reduce the processor/space bounds at the expense of having a larger time bound. For example, if we choose $k = \log^2 n$ (instead of $\log n$), then the time bound increases to $O(\log^3 n)$ while the processor (resp., space) bound decreases to $O(n^2/\log^5 n)$ (resp., $O(n^2/\log^2 n)$). This is mainly due to Step 3 of the algorithm. The *time* $\times$ *processors* product of the algorithm with $k = \log^2 n$ is hence a factor of $\log n$ smaller than the one with $k = \log n$.

References

[1] A. Aggarwal and J. Park. Notes on searching in multidimensional monotone arrays. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 497–512. IEEE, 24–26 Oct. 1988.

[2] A. Apostolico, M. J. Atallah, L. L. Larmore, and S. McFaddin. Efficient parallel algorithms for string editing and related problems. *SIAM J. Comput.*, 19(5):968–988, Oct. 1990.

[3] M. J. Atallah and D. Z. Chen. Computing the all-pairs longest chains in the plane. *International Journal of Computational Geometry & Applications*, 5(3):257–271, 1995.

[4] M. J. Atallah, M. T. Goodrich, and K. Ramaiyer. Biased finger trees and three-dimensional layers of maxima. In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, pages 150–159. ACM, 6–8 June 1994.

[5] M. J. Atallah and S. R. Kosaraju. An efficient algorithm for maxdominance, with applications. *Algorithmica*, 4:221–236, 1989.

[6] M. J. Atallah and S. R. Kosaraju. An efficient parallel algorithm for the row minima of a totally monotone matrix. *Journal of Algorithms*, 13(3):394–413, Sept. 1992.

[7] G. Birkhoff. Lattice theory. *American Mathematical Society Colloquium Publications*, 25, 1979.

[8] E. Cohen. Efficient parallel shortest-paths in digraphs with a separator decomposition. *Journal of Algorithms*, 21(2):331–357, Sept. 1996.

[9] R. Cole. Parallel merge sort. *SIAM J. Comput.*, 1(4):770–785, Aug. 1988.

[10] R. B. K. Dewar, S. M. Merritt, and M. Sharir. Some modified algorithms for Dijkstra's longest upsequence problem. *Acta Inf.*, 18(1):1–15, 1982.

[11] E. W. Dijkstra. Some beautiful arguments using mathematical induction. *Acta Inf.*, 13(1):1–8, 1980.

[12] M. L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, pages 29–35, 1975.

[13] Y. Han, V. Y. Pan, and J. H. Reif. Efficient parallel algorithms for computing all pair shortest paths in directed graphs. *Algorithmica*, 17(4):399–415, Apr. 1997.

[14] J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, Reading, Massachusetts, 1992.

[15] T. Kameda. On the vector representation of the reachability in planar directed graphs. *Inf. Process. Lett.*, 3(3):75–77, Jan. 1975.

[16] D. Kelly and I. Rival. Planar lattices. *Canadian Journal of Mathematics*, 27(3):636–665, June 1975.

[17] P. N. Klein and S. Subramanian. A linear-processor polylog-time algorithm for shortest paths in planar graphs. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 259–270. IEEE, 3–5 Nov. 1993.

[18] A. Lempel, E. S., and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs, International Symposium*, pages 215–232, 1966.

[19] R. Tamassia and J. S. Vitter. Parallel transitive closure and point location in planar structures. *SIAM J. Comput.*, 20(4):708–725, Aug. 1991.

Planar Segment Visibility Graphs

H. Everett* C. T. Hoàng† K. Kilakos‡ M. Noy§

Abstract

Given a set of n disjoint line segments in the plane, the segment visibility graph is the graph whose $2n$ vertices correspond to the endpoints of the line segments and whose edges connect every pair of vertices whose corresponding endpoints can see each other. In this paper we characterize and provide a polynomial time recognition algorithm for planar segment visibility graphs. We use and prove the fact that every segment visibility graph contains K_4 as a subgraph. In fact, we prove a stronger result: every set of n line segments admits at least $n-3$ empty convex quadrilaterals.

1 Introduction

Visibility graphs have been defined for many classes of objects and types of visibility [12]. In this paper we consider the visibility graphs of line segments in the plane. Given a set S of n disjoint line segments in the plane, the *segment visibility graph* of S, denoted G_S, is the graph whose $2n$ vertices correspond to the endpoints of the line segments and whose edges connect every pair of vertices whose corresponding endpoints can see each other; two segment endpoints can see each other if they form a segment or if the line segment connecting them intersects no other segment. Efficient algorithms exist for constructing such visibility graphs [6]. On the other hand, despite much effort, no efficient algorithms are known for recognizing segment visibility graphs; that is, for determining whether or not a given graph is a segment visibility graph [11]. The problem has been solved for certain special cases including that of whole segment visibility graphs with uni-directional visibility in which two segments are visible if there is some point on one segment that can see some point on the other segment in the given direction [1, 13, 14].

In this paper we determine which segment visibility graphs are planar graphs. Our characterization leads to a polynomial-time recognition algorithm for this class. This is not the first time that the restriction to planar graphs has yielded interesting results; see [10] for results on planar visibility graphs of simple polygons.

The key point in proving our characterization is that planar segment visibility graphs cannot be 4-connected. To prove this claim, we show that every configuration of $n > 3$ disjoint line segments in general position contains K_4 as a subgraph. Actually, we prove a stronger result of independent interest, namely that the $2n$ endpoints determine at least $n-3$ empty convex quadrilaterals, where empty means not containing any portion of a segment. It is worth mentioning that this problem has been studied previously for configurations of point sets instead of segments. Harborth showed [7] that every set of 10 points contains an empty convex pentagon, and Horton [8] constructed for every n a set of n points without empty convex heptagons. The situation for hexagons has not been settled.

Throughout the paper we will consider sets of line segments in *general position*; that is, no three segment endpoints are collinear. Note that this implies that no two endpoints are coincident. The reader is referred to [4] for standard graph theory definitions.

*Département d'Informatique, Université du Québec à Montréal, Case postale 8888, Succ. Centre-Ville, Montréal, Québec, H3C 3P8, Canada, everett@lacim.uqam.ca. This research partially supported by NSERC and FCAR.

†Department of Mathematical Sciences, Lakehead University, 955 Oliver Road, Thunder Bay, P7B 5E1, Canada, chinh.hoang@lakeheadu.ca.

‡Department of Mathematics, London School of Economics, Houghton Street, London, WC2A 2AE, England, kyriakos@winston.lse.ac.uk.

§Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, 5 Pau Gargallo, 08028 Barcelona, Spain, noy@ma2.upc.es.

2 Empty convex quadrilaterals

Let S be a set of n disjoint line segments. By an *empty convex quadrilateral* in S we mean four endpoints of S that are the vertices of a convex quadrilateral C such that no segment intersects the interior of C. The next result, besides its application to the next section, is of independent interest.

Theorem 2.1 *A set of n disjoint line segments always contains at least $n-3$ empty convex quadrilaterals.*

Proof. Construct a partition of the plane as follows. Extend every segment in both directions until it hits another segment or a previously drawn extension. The resulting partition depends of course on the order in which the segments have been extended but, as Figure 1 illustrates, the plane is decomposed into exactly $n+1$ convex regions, some of them unbounded, with disjoint interiors (see [11][p. 259] for a different use of this construction).

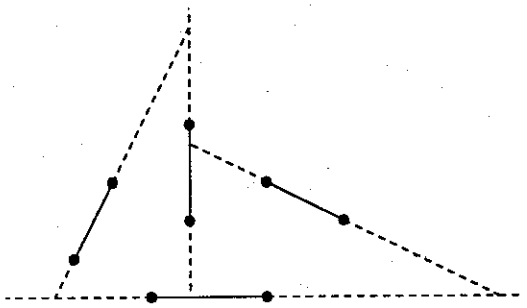


Figure 1: A partition into $n+1$ regions.

Let $m_1, m_2, \ldots, m_{n+1}$ be the number of points in each of these regions. Observe that since the regions are convex and empty, every four points in any of them define an empty convex quadrilateral. Because of the assumption of non-collinearity, every endpoint belongs to exactly two regions. Since there are $2n$ points we have

$$4n = m_1 + m_2 + \cdots + m_{n+1}.$$

The number of empty quadrilaterals is at least

$$\binom{m_1}{4} + \cdots + \binom{m_{n+1}}{4}$$

and it is easy to see that this quantity is minimum, and equal to $n-3$, when four of the m_i are equal to 3 and the remaining m_i equal to 4. □

The example in Figure 2(a) shows 3 segments without empty convex quadrilaterals, and the example in Figure 2(b) shows that the above bound cannot be improved. It also shows that for every n there are sets of n segments containing no empty convex pentagon.

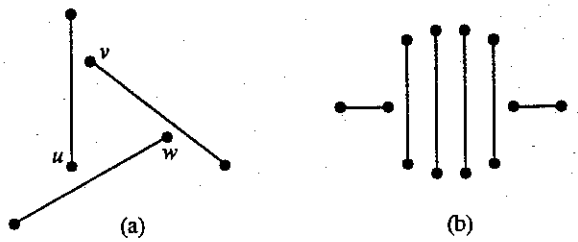


Figure 2: (a) A set of 3 segments; (b) Only $n-3$ empty convex quadrilaterals.

To conclude this section, we strengthen the last result by showing that the number of empty quadrilaterals is proportional to the size of the visibility graph. Our proof is an adaptation of an argument from Bárány and Füredi [3].

Theorem 2.2 *Let S be a collection of n disjoint line segments, and let $G_S = (V, E)$ be its visibility graph. Then among the $2n$ endpoints of S there are at least $\frac{1}{2}(|E| - 6n + 6)$ empty convex quadrilaterals.*

Proof. Let xy be any edge of G_S not a segment. If xy is crossed by another edge zw of G_S, then we can guarantee that xy is a diagonal of an empty convex quadrilateral. Just take the points z' and w' respectively in the triangles xyz and xyw, and closest to xy. Then the convex quadrilateral $xz'yw'$ is empty. This implies that the number of empty convex quadrilaterals is at least half the number of visibility edges crossed by another edge. But the set of non-crossed edges is obviously a plane graph on $2n$ vertices and it has at most $6n-6$ edges; the result follows. □

3 Planar visibility graphs

This section is organized as follows. First we prove some key lemmas regarding the cutsets in segment visibility graphs. In particular, we show that no 4-connected graph is a planar segment visibility graph, with only one exception, and then prove some properties of the 2 and 3-cuts. Next we define a class of planar graphs $\mathcal{D}$ which can be recognized in polynomial time and are segment visibility graphs. Finally we show, using the results on cutsets, that all planar segment visibility graphs belong to $\mathcal{D}$.

We recall that according to Kuratowski's Theorem a graph is planar if, and only if, it does not contain a subdivision of K_5 or $K_{3,3}$ [4]. However, in our proof we only use a much simpler fact, namely that a planar graph cannot contain a subdivision of K_5.

3.1 Cutsets

Our first result is that segment visibility graphs cannot be planar and 4-connected, with the only exception of the visibility graph corresponding to the set F of 3 segments shown in Figure 2(a) (note that G_F is the graph of the octahedron). We remark that the same result has also been shown to hold for visibility graphs of simple polygons [2, 10].

Lemma 3.1 *The only 4-connected planar segment visibility graph is G_F.*

Proof. We make use of a corollary of Menger's Theorem [4], the so called Fan Lemma: if a given graph is k-connected and X is a proper subset of its vertices with $|X| \geq k$, then for any vertex v not in X, there are k internally disjoint paths that link v to distinct vertices of X.

If now S is a set of more than 3 segments, by Theorem 2.1, G_S has a subgraph H isomorphic to K_4. Applying the Fan Lemma to $X = V(H)$ and any vertex v of $V(G_S) - V(H)$, we obtain a graph homeomorphic to K_5, showing that G_S is non planar. When $|S| \leq 3$, the result can be checked by inspection. □

Recall that a *cutset* in a graph is a minimal set of vertices whose removal disconnects the graph. A k-cut is a cutset of size k. The previous lemma tells us that if a segment visibility graph is planar then it has no cutset of size 4 or greater. To characterize the cutsets of size less than 4 we make use of a triangulation of the line segments: a triangulation of S is a planar graph whose vertices are the endpoints of the segments of S and whose edges consist of S plus a set of non-crossing visibility edges, which, when added to S, partition the interior of the convex hull of S into triangles. Every set of disjoint line segments admits such a triangulation.

Now, a cutset of G_S must also be a cutset of any triangulation of S (although the converse is not necessarily true). Since triangulations are 2-connected, G_S contains no cut vertex. Laumond has characterized the cutsets of size less than 4 in a plane triangulation [9]; we follow the description in [5]. A *chord* of a triangulation is an edge connecting two nonconsecutive vertices of the outer face. A *complex triangle* is a triangle that does not form the boundary of a face. A path consisting of an interior vertex connected to two non-consecutive vertices on the exterior face is called a *separating path of length 3.*

Lemma 3.2 ([9]) *A triangulation is*

1. *3-connected if and only if it does not have a chord,*
2. *4-connected if and only if it does not have a chord, a complex triangle or a separating path of length 3.*

We now show that in a segment visibility graph, a 2-cut must be a chord which is also a segment.

Lemma 3.3 *A 2-cut of a segment visibility graph is a segment whose endpoints are nonconsecutive on the convex hull.*

Proof. Clearly, a segment whose endpoints are nonconsecutive on the convex hull is a 2-cut. Let $\{u, v\}$ be a 2-cut of a segment visibility graph G_S and suppose uv is not a segment. By Lemma 3.2, uv is an edge of G_S and u and v are nonconsecutive on the convex hull of S. Let a and b be the two vertices adjacent to u and v in some triangulation of S. Notice that a and b are in different connected components of $G_S - \{u, v\}$. Now $aubv$ is a convex quadrilateral with no segment endpoint in its interior. No segment intersects au, ub, bv or av since these are visibility edges. Thus no segment intersects ab, so a and b are visible which is a contradiction. □

Our last lemma shows that a separating path in a triangulation of a set of segments necessarily contains one of the segments.

Lemma 3.4 *Let $|S| \geq 3$ and let Q be a separating path of length 3 in a triangulation T of S. Then the subgraph of G_S induced by Q contains a segment of S.*

Proof. Let $Q = \{u, w, v\}$ be a separating path of T with w the interior vertex. Both $\{u, w\}$ and $\{w, v\}$ are edges of G_S and we need to prove that one of these is a segment; suppose this is not the case. Since G_S is 2-connected, by Lemma 3.3, uv is not a segment. $G_S - Q$ consists of exactly two components, say C_1 and C_2. Consider the face f created by the deletion of w from T. Let $u, a_1, \ldots, a_k, v, b_1, \ldots, b_l$ be the vertices of f, in counterclockwise order where $\{a_1, \ldots, a_k\} \subseteq C_1$ and $\{b_1, \ldots, b_l\} \subseteq C_2$. If uvw is a face of T then one of $\{a_1, \ldots, a_k\}$ and $\{b_1, \ldots, b_l\}$ is empty; if $\{a_1, \ldots, a_k\}$ is empty then take a_1 to be the vertex adjacent to uv in C_1 and if $\{b_1, \ldots, b_l\}$ is empty then take b_1 to be the vertex adjacent to uv in C_2. Excatly one of ww', $w' \in \{a_1, \ldots, a_k, b_1, \ldots, b_l\}$, is a segment. The interior of f contains no segment endpoint except w and no segment crosses any of its edges. Now it's easy to see that some vertex of $\{a_1, \ldots, a_k\}$ sees some vertex of $\{b_1, \ldots, b_l\}$ which is a contradiction. □

3.2 The class $\mathcal{D}$

The members of $\mathcal{D}$ are precisely the special graph G_F plus those graphs G that can be obtained from a set of edges $\{v_1w_1, \cdots, v_nw_n\}$ as follows. Each v_i, w_i, $1 \leq i < n$, is joined to v_{i+1} and w_{i+1}. In addition, if $v_{i-1}v_{i+1} \notin E(G)$, v_iv_{i+2} could be an edge of G and if $w_{i-1}w_{i+1} \notin E(G)$, w_iw_{i+2} could be an edge of G, $1 < i < n-1$. However, it cannot be that both v_iv_{i+2} and w_iw_{i+2} are edges of G. The edges of the form v_iw_i are the *links* of G. The edges v_1w_1 and v_nw_n are its *end links*. Two links v_iw_i, v_jw_j are *adjacent* if $|i - j| = 1$.

Note that each 3-connected member of $\mathcal{D} \backslash \{G_F\}$ on $2n$ vertices is isomorphic to the graph shown in Figure 3. This graph is maximal planar (i.e. a triangulation) for one can embed triangle $v_1w_1w_2$ on the plane, then v_2 inside this triangle, v_3 inside triangle $v_1v_2w_2$ and so on. Now, since the only 2-cuts in a member of $\mathcal{D}$ are links, $\mathcal{D}$ consists entirely of planar graphs.

Note also that every graph in $\mathcal{D}$ is actually a segment visibility graph; the 3-connected components can be embedded much like in Figure 3.

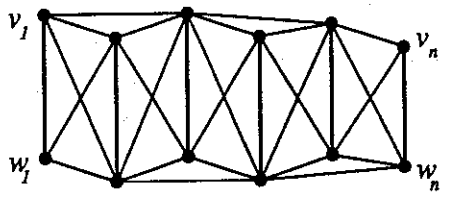


Figure 3: A 3-connected member of $\mathcal{D}$.

3.3 The Main Theorem

Theorem 3.5 *$\mathcal{D}$ is the class of planar segment visibility graphs.*

Proof. Let S be a set of line segments. We shall show by induction on $|S|$ that G_S belongs to $\mathcal{D}$ and that its links correspond to segments of S, like in Figure 3. The case in which G_S is not 3-connected, and therefore 2-connected, can be easily handled with a decomposition argument based on the separating edge described in Lemma 3.3. So we assume that G_S is 3-connected.

Let r be the number of points in the convex hull CH of the $2n$ endpoints and let T be a triangulation of S. By Euler's formula, T has $6n - 3 - r$ edges. Now by Theorem 2.1, S contains at least $n - 3$ empty convex quadrilaterals. Since the two diagonals of a convex quadrilateral cannot both be in T, this gives $n-3$ additional edges not in T. Hence G_S has at least $7n-6-r$ edges. But since the graph is planar it must be $7n - 6 - r \leq 6n - 6$, that is $r \geq n$. Then either (1) a segment $s = uv$ has its two endpoints in CH, or (2) every segment has one endpoint in CH. We treat the two cases separately.

(1) In this case s has to be an edge of the convex hull, otherwise it would be a 2-cut. Then $G_{S \backslash s}$ is planar and 3-connected and by induction $G_{S \backslash s}$ is in $\mathcal{D}$. Considering the different positions where s can be placed and still produce a planar graph it is clear that G_s is also in $\mathcal{D}$ and that s is an end link of G_S.

(2) Since G_S is 3-connected there is a 3-cut $Q = \{a, b, c\}$. It has to be also a cut of the triangulation, but since every segment has one endpoint in the convex hull, it cannot be a separating triangle, hence it has to be a separating path. (It is worth noticing at this point that not all 3-cuts in visibility graphs contain a segment: one can place segments inside the triangle uvw in Figure 2(a) so that $\{u, v, w\}$ becomes a 3-cut.)

By Lemma 3.4, Q contains a segment. Say $s = ab$ is the segment, a and c are in CH, and d is the other endpoint in the segment containing c. Let l_{ab} be the line thorugh a and b. Then l_{ab} cannot be intersected by any segment but cd, since otherwise Q would not be a cut. If l_{ab} intersects cd, it can be shown that G_S contains a subdivision of K_5 (an example is shown in Figure 4). Thus l_{ab} divides the set of segments into two disjoint sets S' and S'', one including ab and the other including cd. By induction both $G_{S'}$ and $G_{S''}$ are in $\mathcal{D}$, and a discussion of how they can be glued and still produce a planar graph shows that G_S is in $\mathcal{D}$. □

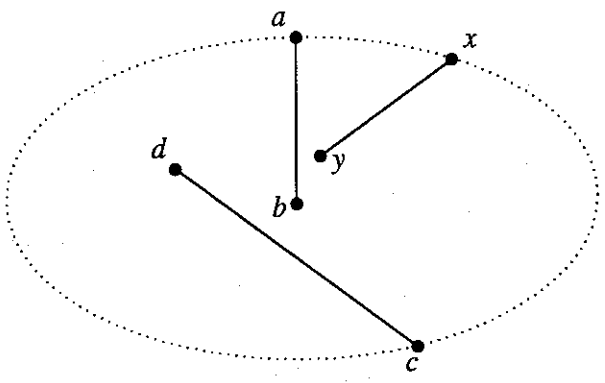


Figure 4: Vertices $abcxy$ produce a subdivision of K_5.

It is easy to see that the class $\mathcal{D}$ can be recognized in polynomial time. Thus we have the following:

Corollary 3.6 *Planar segment visibility graphs can be recognized in polynomial time.*

References

[1] T. Andreae, Some results on visibility graphs, *Discrete Applied Math.* 40 (1992), 5-18.

[2] J. Abello, H. Lin and S. Pisupati, On visibility graphs of simple polygons, *Congressus Numerantium* 90 (1992), 119-128.

[3] I. Barany and Z. Füredi, Empty simplices in Euclidean space, *Canad. Math. Bull.* 30 (1987), 436-445.

[4] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, MacMillan Co., New York (1976).

[5] T.K. Dey, M.B. Dillencourt and S.K. Ghosh, Triangulating with High Connectivity, in *Proc. of the 6th Canadian Conf. on Computational Geometry* (199x), 339-343.

[6] S.K. Ghosh and D.M. Mount, An output-sensitive algorithm for computing visibility graphs, *SIAM J. Comput.* 20 (1991), 888-910.

[7] H. Harborth, Konvexe Fünfecke in ebenen Punktmengen, *Elem. Math.* 33 (1978), 116-118.

[8] J.D. Horton, Sets with no empty convex 7-gon, *Canad. Math. Bull.* 26 (1983), 482-484.

[9] J.-P. Laumond, Connectivity of plane triangulations, *Information Processing Letters* 34 (1990), 87-96.

[10] S.-Y. Lin and C. Chen, Planar visibility graphs, in *Proc. of the 6th Canadian Conf. on Computational Geometry* (1994), 30-35.

[11] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, New York (1987).

[12] J. O'Rourke, Computational Geometry Column 18, *SIGACT News* 24 (1993), 20-25.

[13] R. Tamassia and I. Tollis, A Unified Approach to Visibility Representations of Planar Graphs, *Discrete and Computational Geometry* 1 (1986), 321-341.

[14] S. Wismath, Characterizing bar line-of-sight graphs, in *Proc. 1st ACM Symp. on Computational Geometry* (1985) 147-152.

The Visibility Graph Contains a Bounded-Degree Spanner

Gautam Das*

Abstract

Given a collection of polygonal obstacles with n vertices on the place, and any $t > 1$, we present an $O(n \log n)$ time algorithm that constructs a bounded-degree t-spanner of the visibility graph, without first having to construct the visibility graph.

1 Introduction

An *Euclidean graph* is defined as a graph whose vertices are points in k-dimensional space, edges are line segments joining pairs of points, and edge weights are from the underlying distance metric, typically the L_2 metric. If all edges are present, the graph is a *complete Euclidean graph*, otherwise it is a *non-complete Euclidean graph*. A well-known example of a non-complete Euclidean graph is the *visibility graph*, defined as follows. Consider a scenario where we are given a collection of pairwise disjoint polygons on the plane. This frequently arises in motion planning problems, where the polygons represent obstacles in a cluttered workspace within which a point robot has to navigate. Consider a graph over the polygon vertices, where an edge (u, v) belongs to the graph if the corresponding line segment does not intersect the interior of any obstacle. Such a graph is known as the *visibility graph*. This graph is useful because it contains the shortest obstacle-avoiding path between any pair of vertices. Visibility graphs have been the subject of a great deal of recent research, from both computational and combinatorial aspects (for example, see [14, 16, 17, 22]). It is known that while visibility graphs are not necessarily complete, they can be quite dense, with as many as $\Omega(n^2)$ edges and $\Omega(n)$ degree. It is of interest to investigate whether a visibility graph contains a sparse subgraph which "approximates" shortest paths between all pairs of vertices. Such a subgraph would be a more compact structure in motion planning applications, or in applications where a communication network is being designed (for example, a road network linking all the vertices).

We make this notion of "approximate" shortest paths more precise. Let $G = (V, E)$ be a n-vertex connected graph with positive edge weights. A subgraph G' is a *t-spanner* if for all $u, v \in V$, the distance between u and v in the subgraph is at most t times the corresponding distance in G. The value t is known as the *stretch factor* of the spanner. Spanners are important structures since they represent the original graph more compactly, albeit approximately. In constructing spanners, it is frequently necessary to endow them with additional properties, such as few edges, small total weight, small degree, small diameter, etc. Spanners of arbitrary weighted graphs as well as special classes of graphs such as complete Euclidean graphs have been the subject of much recent research. Spanners find applications in a variety of areas: communication network design, distributed algorithms, network routing, computational geometry and robotics. They are also fascinating from a theoretical point of view, and possess many interesting combinatorial and geometric properties. A good bibliography of past spanner research may be found in [5]. Additionally, in this paper we list several recent references.

While spanners of complete Euclidean graphs have been well studied, relatively little work has been accomplished on non-complete Euclidean graphs, such as for example, visibility graphs. An early result is by Clarkson ([6]) who showed how to construct, for any $t > 1$, a linear-sized t-spanner of the visibility graph in $O(n \log n)$ time without having to first construct the visibility graph. Clarkson (and later Chen, [7]) applied this spanner to solving approximate shortest path problems. In [8], Chew shows that the *constrained Delaunay triangulation* is a planar $O(1)$-spanner of the visibility graph, and can be constructed in $O(n \log n)$ time (however, in this result the stretch factor cannot be arbitrarily close to 1). Recently an $O(n \log n)$ time algorithm has been designed by Arikati et. al. ([2]) to construct, for any $t > 1$, a *Steiner* t-spanner (here the spanner is not strictly a subgraph of the visibility graph because it may contain additional Steiner vertices and edges, however distances be-

*Dept. of Mathematical Sci., The Univ. of Memphis, Memphis, TN 38152, USA, dasg@next1.msci.memphis.edu

tween obstacle vertices still stretch by at most t). These Steiner spanners find applications in answering all-pairs shortest path queries amidst obstacles.

But suppose we are interested in constructing a t-spanner such that, (a) it is a subgraph of the visibility graph, and (b) it has *bounded degree*? The problem is interesting from a theoretical standpoint, because we are trying to discover new combinatorial and geometric properties of visibility graphs and their subgraphs. From a practical standpoint, such a spanner may be used in the design of a road network linking all obstacle vertices, where the objective is to decrease congestion by only allowing a few links to be incident to any vertex. We mention that the corresponding bounded-degree spanner problem for complete Euclidean graphs in k-dimensional space has attracted considerable attention recently (for example, see [3, 5, 9, 20]). However bounded-degree spanners of visibility graphs seem harder to construct, mainly because previously developed techniques for complete Euclidean graphs cannot be immediately used (such techniques rely on the fact that "any vertex can be joined with any other vertex", which is not true when there are obstacles).

In this paper we have developed an algorithm for constructing bounded-degree spanners of visibility graphs. Our algorithm combines a few old techniques with several new techniques. For example, the algorithm is loosely based on the "covering by cones" paradigm, which in the past has been very useful in spanner construction (see [1, 6, 18]). What is interesting is that we extend the idea much beyond its earlier scope, for example when we have to deal with the special geometric constraints that the polygonal obstacles pose. The following theorem summarizes our result.

Theorem 1.1 *Given a set of polygonal obstacles with n vertices in the plane, and any $t > 1$, a bounded-degree t-spanner of the visibility graph exists, and can be constructed in $O(n \log n)$ time. The constants implicit in the big-O depend on t.*

The rest of the paper is organized as follows. In Section 2 we review Clarkson's spanner (see [6]), because it provides the foundation for our spanner algorithm. In Section 3 we show how to construct a bounded-degree spanner, thereby proving Theorem 1.1. We present some open problems in Section 4.

2 Clarkson's Spanner

In this section we start by reviewing Clarkson's spanner (see [6]), which has linear size, but may not have bounded degree. (In our presentation, we retain the main ideas, but present the algorithm somewhat differently. For example, we use a plane sweep, and also use the concept of *projected distances*).

Consider an infinite horizontal line passing through an arbitrary point z. Let θ be a small constant angle (its actual value depends on the given t) and let $L_1, L_2, \ldots, L_{\pi/2\theta-1}$ be semi-infinite rays radiating downward from z such that the angle between adjacent rays is θ. This partitions the lower half-plane into a constant number of unbounded triangles called *cones*, $C_1, C_2, \ldots, C_{\pi/2\theta}$. For each cone C_i, define the *axis* R_i as the semi-infinite ray from z which angularly bisects the cone. Let y be any other point in the interior of C_i. The *projected distance* between z and y, $proj(z, y)$, is defined to be the distance between z and the projection of y on R_i. For a small θ, clearly the projected distance is almost equal to the *actual distance*, $d(z, y)$. (Projected distances were first used in [18] for constructing spanners).

The algorithm sweeps the plane in a particular direction (say from bottom to top, to be consistent with the diagrams to be introduced later), and on encountering a vertex v, decides to select only some of the visibility graph edges that connect it to the vertices below. The selection of visibility graph edges is simple. Translate all the cones $C_1, C_2, \ldots, C_{\pi/2\theta}$ such that their apexes become v. For every cone C_i, of all the visibility graph edges incident to v and contained within C_i, the algorithm selects the edge with the shortest projected length. Once the sweep is over, the selected edges represent the spanner, G.

It is easy to see that the output has a linear number of edges, since at every vertex at most a constant number of edges are selected (at most one per cone). However, the degree may not be a constant. To see this, imagine that the algorithm is actually creating a *directed* graph; at vertex v the edges that are selected are given downward directions (from v to the other endpoints below). While the output graph has a bounded out-degree, it may have an unbounded in-degree. The output is also a t-spanner of the visibility graph; for a proof we refer the reader to [6]. The algorithm can be implemented to run in $O(n \log n)$ time without having to first create the visibility graph, using techniques such as planar point location and *conical Voronoi*

diagrams; the details are in [6].

3 Bounding the Degree

In this section we describe our more complex algorithm, which produces a bounded-degree spanner. The algorithm consists of four steps.

Step 1: *Construction of a linear-sized spanner*:

Select numbers t_1 and t_2 such that $t_1, t_2 > 1$, t_1 is very close to 1, t_2 is somewhat bigger but still small enough so that $t_1 \cdot t_2$ is closer to 1 than to t.[1] Create a t_1-spanner of the visibility graph, using Clarkson's algorithm. Let this spanner be G.

Step 2: *Partition into forests*:

Recall that we had used $O(1)$ cones $C_1, C_2, \ldots, C_{\pi/2\theta}$ in the previous algorithm. We are going to refine this even further. Select an angle α to be much smaller than θ but which divides θ evenly (some intuition about its value is given in Step 3). We partition each cone C_i into $O(1)$ *subcones* $C_{i,1}, C_{i,2}, \ldots, C_{i,\theta/\alpha}$, such that the angle of each subcone is α. The following terms will be useful in our explanations: the subcones in the central region of C_i are known as *central subcones*, while the subcones to the far left or far right of C_i are known as *peripheral subcones*. Notice that the total number of subcones over all the cones is $\pi/2\alpha$, which is a constant.

We partition G into a constant number of subgraphs (actually forests) as follows. Consider G as a directed graph, and let (v, u) be any directed edge of G (i.e. directed downwards from v to u). If it lies inside the subcone $C_{i,j}$ (with apex at v), then (v, u) belongs to the graph $G_{i,j}$. It is easy to see that each $G_{i,j}$ has an out-degree of one, but may have an unbounded in-degree. Clearly the undirected version of $G_{i,j}$ is a forest.

Step 3: *Removal of more edges*:

In this step, the algorithm performs a downward sweep of G (actually over all the $G_{i,j}$'s simultaneously), and at each vertex possibly removes some of the incoming edges. The output graph, G', will be a spanner of the visibility graph. It will have a smaller, though still not constant, in-degree. The stretch factor will be larger than that of G. The logic for selection or removal of edges of $G_{i,j}$ is as follows. Let c be a constant (approximately) equal to $\frac{t_1 \cdot t_2 - t_2}{t_1 \cdot t_2 - 1}$ (the exact value of c is somewhat different; it also depends on α and θ, but we omit details from this version). Notice that $0 < c < 1$. Since t_1 is selected very close to 1, c is very close to 0. Suppose the downward sweep visits a vertex v. Sort the incoming edges of $G_{i,j}$ at v by increasing length. Select the shortest, whose length is say l_1. Remove all edges of length at most l_1/c. Select the smallest of the remaining, say l_2. Remove all edges of length at most l_2/c. Repeat this process until all incoming edges are either selected or removed. Do this for all the forests, then sweep downwards to the next vertex.

Once the sweep is over, each forest $G_{i,j}$ has been reduced to a sparser forest, say $G'_{i,j}$. The union of these forests, G', is the output of Step 3.

Before we describe Step 4, let us analyze G', the output of Step 3. Clearly the in-degree of any $G'_{i,j}$ may not be bounded. However if two incoming edges of $G'_{i,j}$ meet at a common vertex, then one edge is much longer than the other. In addition, the following lemma shows that G' is a spanner.

Lemma 3.1 *G' is a $(t_1 \cdot t_2)$-spanner of the visibility graph.*

Proof : In this version we use very rough calculations and simply sketch the proof. Essentially we have to show that G' is a t_2-spanner of G. Suppose an edge of G, say (u, v), is removed while the algorithm is visiting v. There should be an alternate path in G' of length at most $t_2 \cdot d(u, v)$. Suppose (u, v) originally belonged to some $G_{i,j}$. Then some other edge, (w, v) belongs to $G'_{i,j}$ such that $d(u, v) < d(w, v)/c$. Let us use Figure 1 as an illustration. For both u and w, the position of their C_i cone is shown, where the right ray of u's cone intersects the left ray of v's cone at x. It is not hard to see that the line segments (u, x) and (x, w) *do not intersect other obstacles*. We also observe that the two edges (u, v) and (w, v) are almost parallel, since α is very small. Finally the angle uxw is θ, and since α is much smaller than θ, the length of (x, w) is negligible compared to the length of (v, w).

Define $cv(w, x, u)$ to be the convex chain obtained by placing a stretched rubber band from w to x to

[1] Due to lack of space, we will avoid deriving the exact values of the various constants used in this version of the paper

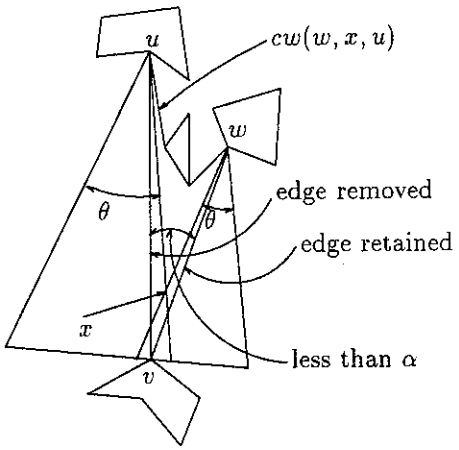


Figure 1: Analysis of Step 3

u, anchoring it at w and u and releasing it at x. It will assume a convex shape because the shrinkage will be halted by various obstacles. The chain is confined within the triangle wxu, where (w, u) is defined as the *base boundary* and (w, x) and (x, u) are defined as the *side boundaries* of the chain. Note that this chain is a path from w to u in the original visibility graph. By an induction argument which we omit, it can be assumed that there is a path P from w to u in G' of length at most $t_1 \cdot t_2$ times the length of $cv(w, x, u)$. We thus have an alternate path Q from v to u in G' as follows: go from v to w, then go along P from w to v. The length of Q is at most $d(v, w) + t_1 \cdot t_2 \cdot (d(w, x) + d(x, u))$. But since (v, u) and (v, w) are almost parallel, and $d(x, w)$ is negligible compared to (v, w), the length of Q is maximized when $d(u, w)$ is the smallest possible, i.e. when $d(u, w) = c \cdot d(v, u)$. In this situation $d(u, x) + d(x, v)$ is approximately equal to $(1 - c) \cdot d(u, v)$. Substituting these quantities in, we get the length of Q to be approximately at most $c \cdot d(v, u) + t_1 \cdot t_2 \cdot (1 - c) \cdot d(v, u)$. Substituting for c, this simplifies to at most $t_2 \cdot d(u, v)$. Thus we can conclude that G' is a t_2-spanner of G, hence a $(t_1 \cdot t_2)$-spanner of the visibility graph. ■

At this stage we can make some more observations. Consider a $G_{i,j}$ whose corresponding subcone is $C_{i,j}$. Let v be a vertex with a large in-degree. If $C_{i,j}$ is a central subcone, then all the incoming edges at v will be approximately of the same length (since α is much smaller than θ). In this case, Step 3 will remove all but the shortest edge. On the other hand, if $C_{i,j}$ is a peripheral subcone, then the incoming edges at v could be of all possible lengths.

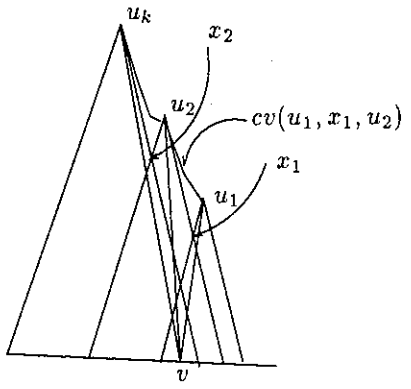


Figure 2: Constructing the bounded-degree spanner

Though Step 3 prunes some of them, an unbounded number could be left over. We can only guarantee that for any pair of left over edges, the shorter edge is much shorter (at most c times) than the longer edge. Figure 2 (or a symmetric equivalent) correctly describes the geometry of all incoming edges of $G'_{i,j}$ at v. Notice that the order of these edges by length is the same as their order by slope.

Step 4: *Achieving bounded degree*:

> This final step is different from the others because here we remove more edges, and also *add back* some edges. The idea is to consider each $G'_{i,j}$, and create another forest $G''_{i,j}$ which has bounded degree. However, $G''_{i,j}$ is not necessarily a subgraph of $G'_{i,j}$. The union of all $G''_{i,j}$, defined as G'', is the final output of our algorithm.
>
> The details of the construction of G'' are simple. For all $G'_{i,j}$, for all vertex v of $G'_{i,j}$, perform the following operation. Let $(u_1, v), (u_2, v), \ldots, (u_k, v)$ be the incoming edges at v in increasing length (see Figure 2). Retain (u_1, v). Remove $(u_2, v), (u_3, v), \ldots, (u_r, v)$. *Add* the chains $cv(u_1, x_1, u_2), cv(u_2, x_2, u_3), \ldots, cv(u_{k-1}, x_{k-1}, u_k)$.

This completes the description of the algorithm. We now analyze the algorithm and its output G''.

Lemma 3.2 *G'' is a t-spanner of the visibility graph.*

Proof : We sketch the proof. Consider any edge (u_m, v) of G' that got removed. Consider the alternate path in G'': go from v to u_1, then along $cv(u_1, x_1, u_2)$, then along $cv(u_2, x_2, u_3)$, and so on

until you reach u_m. Since α is small, and the constant c (used in Step 3) is small, from the geometry of the situation it can be shown that this path is not much longer than $d(u_m, v)$. In fact, by selecting smaller α and smaller c (which can be achieved by selecting t_1 closer to 1), we can make this path length as close to $d(u_m, v)$ as we like. Since G' is a $(t_1 \cdot t_2)$-spanner of the visibility graph, it follows that G'' is also a spanner of the visibility graph, but with a slightly larger stretch factor. Since $t_1 \cdot t_2$ has been selected to be closer to 1 than to t, we can make sure that the stretch factor of G'' is no more than t. ■

The next two lemmas eventually show that the degree of G'' is bounded. In the construction of $G''_{i,j}$, recall that at a vertex v, all incoming edges except the shortest are removed, and several convex chains are added instead. For each such convex chain, we define its *base vertex* to be v.

Lemma 3.3 *If we start with the empty plane, and draw on it all the convex chains of $G''_{i,j}$ and their respective side boundaries, all the line segments in the arrangement will be pairwise disjoint, except possibly for sharing common endpoints.*

Proof : Let $cv_1 = cv(u_m, x_m, u_{m+1})$ and $cv_2 = cv(u_p, x_p, u_{p+1})$ be any two convex chains of $G''_{i,j}$. If they have the same base vertex, v, then the lemma is clearly true (as Figure 2 shows). On the other hand, suppose the two chains have different base vertices, say v_1 and v_2 respectively. This involves a detailed case analysis, which we omit. Instead, consider Figure 3 which shows the most "crowded" situation where we try and make the two chains intersect each other. However, since v_1 is an obstacle vertex, the convex chain cv_2 either includes v_1 or is below it. Thus it neither intersects cv_1 nor the side boundaries of cv_1. ■

Lemma 3.4 *G'' has bounded degree.*

Proof : Since there a constant number of $G''_{i,j}$ it will suffice to prove that each of the latter has bounded degree. Consider any $G''_{i,j}$. It has two kinds of edges, the convex chains, and the shortest edges retained from $G'_{i,j}$ at every vertex. Consider the subgraph consisting of the convex chains. Due to Lemma 3.3, its degree is at most 2. Next consider the subgraph consisting of the shortest edges. The degree of this subgraph is at most 2, because at any vertex there is at most one incoming edge and at most one outgoing edge. Thus the degree of $G''_{i,j}$ is bounded, which implies that the degree of G'' is also bounded. ■

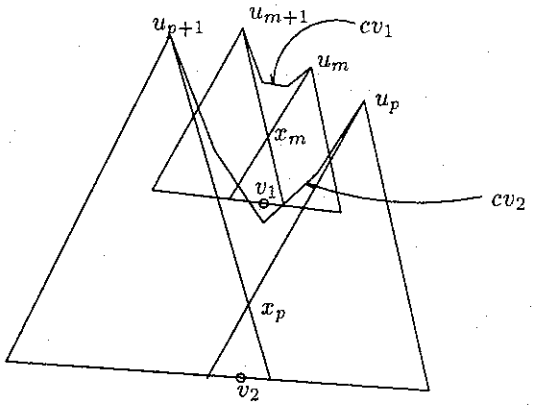


Figure 3: Convex chains and side boundaries are disjoint

We now present an efficient implementation of the algorithm. Step 1 takes $O(n \log n)$ time. Step 2 takes $O(n)$ time since there are $O(n)$ edges in G and assigning an edge to its respective forest takes $O(1)$ time. Step 3 involves sorting and thus takes $O(n \log n)$ time per $G_{i,j}$, thus $O(n \log n)$ time overall.

Implementing Step 4 is nontrivial. For each forest $G'_{i,j}$ the main task is to compute several convex chains. Let the complete set of convex chains to be constructed for $G'_{i,j}$ be $CV_{i,j}$. First start with the empty plane, and lay out the convex hull of the original set of obstacles. Then for each chain in $CV_{i,j}$, lay out its two side boundaries on the plane (the two side boundaries are known, even if the chain is not yet computed). This will result in a collection of line segments in the interior of the hull, pairwise disjoint except at endpoints (this is due to Lemma 3.3). Treat these line segments as edge obstacles, and compute a trapezoidal decomposition of the interior of the hull, where the parallel boundaries of the trapezoids are parallel to the axis of the cone C_i. Using planar point location, assign each original obstacle vertex to the trapezoid that contains it. For each chain cv_r, let the set of obstacle vertices inside the trapezoids immediately above the chain's side boundaries be V_r. Using an optimal convex hull algorithm, compute cv_r *using only* V_r as input. It should be clear that for two different chains cv_r and cv_s, the corresponding sets V_r and V_s are disjoint. Thus computing $G''_{i,j}$ takes $O(n \log n)$ time, and therefore Step 4 takes $O(n \log n)$ time. Thus the entire algorithm runs in $O(n \log n)$ time.

4 Open Problems

We conclude this paper with some open problems.

The foremost open problem is, are there are spanners of the visibility graph with a maximum degree of 3? Notice that the spanner in Theorem 1.1 has $O(1)$ degree, but that sheds no light on whether a degree-4 (or even a degree-3) spanner of the visibility graph exists. We mention that degree-3 spanners exist for the case of complete Euclidean graphs without obstacles ([9]). That paper also shows that 3 is a lower bound on the degree. It would be interesting to extend this to the case with obstacles, however we feel this may require considerably more complicated techniques than used in [9].

For complete Euclidean graphs, spanners are known with *several* sparseness properties, in addition to small degree. Some examples are low weight, and small diameter ([4]). Extending these results to the case of spanners of visibility graphs is a challenging problem.

In general, the problem of constructing spanners of non-complete Euclidean graphs (not just visibility graphs) in both two as well as higher dimensions needs to be studied.

References

[1] I. Althöfer and G. Das and D. P. Dobkin and D. A. Joseph and J. Soares: On sparse spanners of weighted graphs: Discrete Comput. Geom., Vol 9, 1993, pp 81–100.

[2] S. Arikati, D. Chen, L. P. Chew, G. Das, M. Smid, C. D. Zaroliagis: Planar spanners and approximate shortest path queries among obstacles in the plane: Proc. European Symp. on Algorithms (ESA), 1996.

[3] S. Arya and M. Smid: Efficient construction of a bounded degree spanner with low weight: Proc. 2nd Annu. European Sympos. Algorithms (ESA), Lecture Notes in Computer Science, Vol 855, 1994, pp 48–59.

[4] S. Arya and G. Das and D. M. Mount and J. S. Salowe and M. Smid: Euclidean spanners: short, thin and lanky: Proc. ACM Sympos. Theory of Comput. (STOC), 1995.

[5] B. Chandra and G. Das and G. Narasimhan and J. Soares: New sparseness results on graph spanners: Proc. 8th Annu. ACM Sympos. Comput. Geom., 1992, pp 192–201.

[6] K. L. Clarkson: Approximation algorithms for shortest path motion planning: Proc. ACM Sympos. Theory of Comput. (STOC), 1987, pp 56–56.

[7] D. Z. Chen: On the all pairs Euclidean short path problem: Proc. SIAM-ACM Sympos. on Discrete Algorithms (SODA), 1995.

[8] L. P. Chew: There are planar graphs almost as good as the complete graph: J. of Computer and System Sciences, 39, 1989, 205–219.

[9] G. Das and P. Heffernan: Constructing degree-3 spanners with other sparseness properties: Intl. Sympos. Algorithms and Comput. (ISAAC), 1993.

[10] G. Das and P. Heffernan and G. Narasimhan: Optimally sparse spanners in 3-dimensional Euclidean space: Proc. 9th Annu. ACM Sympos. Comput. Geom., 1993, pp 53–62.

[11] G. Das and G. Narasimhan and J. S. Salowe: A new way to weigh malnourished Euclidean graphs: Proc. 6th SIAM-ACM Sympos. Discrete Algorithms (SODA), 1995.

[12] G. Das and G. Narasimhan: A fast algorithm for constructing sparse Euclidean spanners: Proc. 10th Annu. ACM Sympos. Comput. Geom., 1994.

[13] D. Eppstein: Spanning Trees and Spanners: Tech. Report 96-16, Dept. of ICS, UC Irvine, 1996.

[14] S. K. Ghosh and D. M. Mount: An output-sensitive algorithm for Computing visibility graphs: SIAM J. Comput. 20, 1991, pp 888–910.

[15] K. Mehlhorn: Data structures and algorithms 1: sorting and searching: Springer-Verlag, 1984, pp 290–296.

[16] M. H. Overmars and E. Wetzel: New methods for computing visibility graphs: Proc. 4th Annu. ACM Sympos. Comput. Geom., 1988, pp 164–171.

[17] M. Pocchiola and G. Vegter: The visibility complex: Proc. 9th Annu. ACM Sympos. Comput. Geom., 1993, pp 328–337.

[18] J. Ruppert and R. Seidel: Approximating the d-dimensional complete Euclidean graph: Proc. 3rd Canad. Conf. Comput. Geom., 1991, pp 207–210.

[19] J. S. Salowe: Construction of multidimensional spanner graphs with applications to minimum spanning trees: Proc. 8th Annu. ACM Sympos. Comput. Geom., 1991, pp 256–261.

[20] J. S. Salowe: On Euclidean spanner graphs with small degree: Proc. 8th Annu. ACM Sympos. Comput. Geom., 1992, pp 186–191.

[21] P. M. Vaidya: A sparse graph almost as good as the complete graph on points in K dimensions: Discrete and Comput. Geom., 6, 1991, pp 369–381.

[22] E. Welzl: Constructing the visibility graph for n line segments in $O(n^2)$ time: Inform. Process. Letters, 20, 1985, pp 167–171.

Contracted Visibility Graphs of Line Segments

J. Bagga[1] S. Dey[2] L. Gewali[3] J. Emert[1] J. McGrew[1]

1. Introduction

Problems dealing with the visibility and connectivity properties of disjoint line segments have been explored by several researchers in recent years [SE87, LM*87, R89, CH91, R92, BE*94]. Tight bounds have been established for the size of the visibility graph induced by line segments in the plane [SE87, CH91]. The (weak) visibility graph induced by a set of vertical line segments is known to be ipo-triangular. Algorithms dealing with the connectivity of line segments can be found in [R89]. In fact, it has been established that the problem of finding a Hamiltonian cycle in the visibility graph of disjoint line segments is NP-hard [R89]. Results dealing with the existence of a Hamiltonian cycle in restricted classes of disjoint line segments are reported in [R92], where it is established that a Hamiltonian cycle always exists in hulled, independent, and unit segments. Very recently, it was demonstrated that the notion of weak visibility is powerful in capturing the geometric information associated with simple polygons [OS97]. Specifically, O'Rourke and Streinu [OS97] have shown that a vertex-edge visibility graph, defined by incorporating weak visibility notion, can be used to construct shortest path trees, visibility polygon, and reflex vertices of a simple polygons. Other interesting results on weak visibility can be found in [W89, S96].

Motivated by the promising usefulness of weak visibility representation, we introduce a new class of visibility graphs, called the contracted visibility graphs (CVG). Contracted visibility graphs are the visibility graphs of disjoint line segments, where visibility edges are formed by using weak visibility between line segments. We establish several properties of CVG which include the following.

- If segments are strongly hulled, then each block in the CVG is complete.
- The neighborhood of degree $d \geq 2$ vertex in a CVG induces at most two components.
- CVG is a tree if and only if it is a path.
- For $n \geq 4$, there can be at most $\lfloor \frac{n}{2} \rfloor$ vertices of degree 1.
- Given an integer k, $n-1 \leq k \leq \binom{n}{2}$, a CVG is realizable with k edges and n vertices.

2. Preliminaries

Consider a set of line segments $S = \{s_1, s_2, ..., s_n\}$ in the plane. The segment endpoint visibility graph (SEVG) of S is the graph whose vertex set V consist of the endpoints of line segments and two vertices are connected to form an edge if the corresponding points are visible to each other (i.e., the line segment connecting them does not intersect with any other segment.)

[1] Department of Computer Science, Ball State University, Muncie, Indiana. email: jay@bsu-cs.bsu.edu, emert@bsu-cs.bsu.edu, mcgrew@bsu-cs.bsu.edu

[2] Sisco Systems Inc CA. email: sdey@sisco.com

[3] Department of Computer Science, University of Nevada, Las Vegas, Nevada. e-mail: laxmi@cs.unlv.edu.

Figure 1a illustrates a SEVG, where solid lines are the line segments and the dashed lines are visibility edges. Note that the line segment themselves are also visibility edges. It has been established that the number of visibility edges in a SEVG induced by n disjoint line segments in at least $5n - 4$ [SE87].

Notions of 'extreme' and 'chordal' line segments have been used to characterize restricted classes of polygons admitting 'nice' connectivity properties [R89,R92]. It may be remarked that a line segment $s_i \in S$ is called *extreme* if it lies on the boundary of the convex hull of S; and it is called *chordal* if it is not extreme and no endpoint in its left half plane is visible to any endpoints in its right half plane. A set of line segments is *completely hulled* if all segments in it are extreme segments. On the other hand, the set S is *strongly hulled* if all line segments in it are not extreme but all endpoints lie on the hull boundary. Similarly, if at least one endpoint of each segment lies on the hull then we get *weakly hulled* segments. Construction of "ordered Hamiltonian circuit" and "half-ordered Hamiltonian circuit" in weakly hulled line segments are considered in [R92].

It is straightforward to show that each segment induces a K_3 (complete graph with three nodes) in SEVG. This is stated in the following observation.

Observation 1: For $n > 1$, each segment is an edge of at least one K_3 in SEVG.

The following lemma about SEVG can be verified easily.

Lemma 1: The degree of every vertex in a SEVG is at least three.

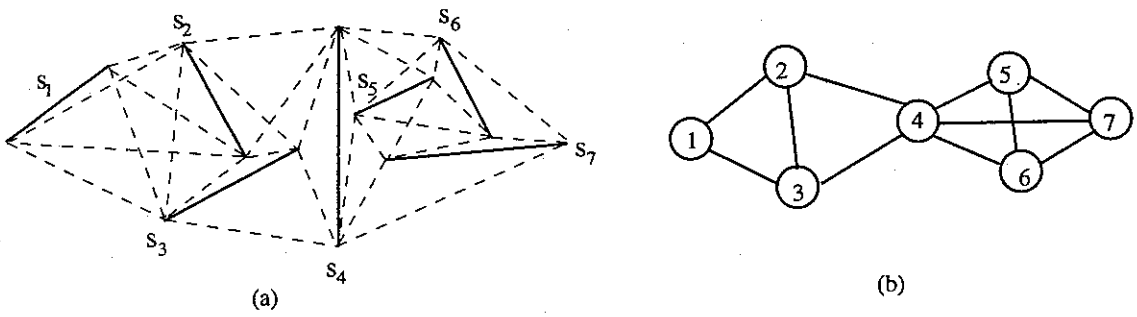


Figure 1: Illustrating SEVG and CVG

2. Contracted Visibilily Graphs

These graphs are defined by using weak visibility between line segments to form edges. Two line segments s_i and s_j are said to be *weakly-visible* to each other if one endpoint of s_i is visible to an endpoint of s_j. (Now onward we will simply use the term 'visible' to indicate 'weakly-visible'.) The *contracted visibility graph (CVG)* of n disjoint line segments $S = \{s_1, s_2, ..., s_n\}$ consist of vertex set V and edge set E such that corresponding to each s_i, a vertex v_i in included in V and two vertices v_i and v_j are connected to form an edge if the corresponding segments s_i and s_j are visible. Figure 1b is the contracted visibility graph of the segments shown in Figure 1a. The term *contracted* refers to the fact that in order to study the visibility representation, two endpoints of a segment are collapsed or contracted to a point.

Lemma 2: If the segments are strongly hulled then each block in the CVG is complete.

Proof: Suppose not. Let us consider a block G in the CVG. In the corresponding set of segments there exists at least one missing edge xy where x and y denote the vertices. Thus in the corresponding set of segments there must exist at least a segment K which blocks the visibility of the segments X and Y. That means the endpoints of X and Y must lie on different half planes

induced by K. However, since the segment K is hulled, its two endpoints lie nonadjacently on the hull and it is thus a chordal segment and is represented by a cut vertex in the corresponding CVG. Hence we get a contradiction that G is not a block. □

Theorem 1: The open neighborhood of any vertex x in a CVG G induces a subgraph with at most two components. In other words, $G - v$ consists of at most two connected components.

Proof: Let $d(x)$ denote the degree of a vertex x in CVG. If $d(x) = 1$ then the theorem is trivially true.

If possible let there be at least *three* components induced by the neighborhood of any vertex x ($d(x) \geq 2$) in a CVG. In the corresponding arrangement of line segments in the plane (Figure 2) let x_u and x_d be the endpoints of the segment represented by x in the CVG. Without loss of generality we might assume that the line segment x is vertical. Since we have assumed that there are at least three components induced by the neighborhood of x in the CVG, the arrangement of segments in the Euclidean plane consists of at least three mutually disjoint sets of line segments S_1, S_2, and S_3 such that any two points α_i, β_j are visible to each other if and only if either of them is x_u or x_d, or α_i, β_j are in the same component. We can view this as the distribution of three disjoint sets in two half planes induced by segment x. By the *pigeon hole principle*, no matter how we distribute the sets, one of the half planes must contain parts or whole of at least two sets. Without loss of generality assume that right half plane contains parts of S_1 and S_2. Now it is a simple matter to verify that at least one line segment in S_1 is visible to a line segment in S_2, implying that S_1 and S_2 are not disjoint - a contradiction. □

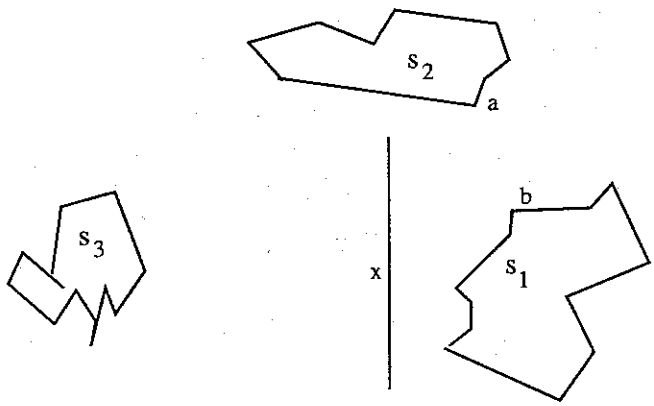


Figure 2: Illustrating the Proof of Theorem 1

Corollary 1: Every cut vertex in a CVG lies in exactly two blocks.

Segments can be arranged so that the CVG becomes a path. However it turns out that if a CVG contains a node with degree three then it can not be a tree. This is stated in the following lemma.

Lemma 3: A CVG is a tree if and only if it is a path.

Proof: Assume to the contrary that CVG can be a tree which is not a path. Hence there exists a vertex v whose degree is 3 or more. Let the vertices adjacent to v be $x_1, x_2, ...x_i$ where $i \geq 3$. However in the CVG, v becomes a cut vertex which lies in 3 or more blocks giving us a contradiction to Corollary 1. Hence if CVG is a tree it must be a path. The converse is trivial. □

Lemma 4: A CVG is either a path or contains at least one K_3.

Proof: If CVG is a path, then we are done. If it is not a path then surely there exists a

segment x which sees more than one segments in one of its half planes (say the left half plane). In such a case if we sweep the left half plane of the segment by a visibility ray originating at one of the endpoints of x, we are guaranteed to find two points a and b which belong to two different segments and are consecutively swept. This implies the existence of a K_3 in the CVG. □

The next lemma establishes the upper bound of the number of degree 1 vertices in a CVG. An example of line segment arrangement can be constructed to show that the stated bound is sharp.

Lemma 5: For $n \geq 4$, there can be at most $\lfloor n/2 \rfloor$ vertices of degree 1 in CVG.
(Proof omitted due to space limitation)

Since CVG is a connected graph it must contain at least $n-1$ edges. Also, the upper bound on the number of edges in CVG is $\binom{n}{2}$. Line segment arrangement examples exist to show that these bounds are tight. In this context it would be interesting to investigate the existence of CVG with the number of edges between the lower bound $n-1$ and the upper bound $\binom{n}{2}$.

Theorem 2: Given a positive integer k, $n-1 \leq k \leq \binom{n}{2}$, a CVG is realizable with k edges and n vertices.

Proof: The proof is by construction and follows an approach similar to the one used in [BE*94]. All segments in the construction are assumed to lie vertically on a horizontal line and are ordered in increasing length from left to right. Also the projection of each (except the rightmost) segment is contained in the segment to its right. We use an induction on n.
Basis: When $n = 2$, $n - 1 = 1 = \binom{n}{2}$ and we have only two segments.
Let us assume that the result holds good for $n-1$ segments. Thus a CVG is realizable having number of edges between $n-2$ and $\binom{n-1}{2}$. Let $\{u_i, l_i\}$ denote the two endpoints of segment s_i. We place a vertical line L to the right of the segment s_{n-1}. For each u_i, $1 \leq i \leq (n-2)$, we draw a *supporting half-line* originating at u_i and passing through u_r $(r > i)$ such that the half-line does not intersect with any other segments. Let these half-lines meet L at points labeled $x_1, x_2, ...x_{n-2}$ where x_i is above x_{i+1} for $1 \leq i \leq n-3$. Let x_0 be a point on L such that x_0 is above x_1. Let the horizontal lines drawn from u_{n-1} and l_{n-1} meet L at x_{n-1} and y_0 respectively. Also let the line between l_{n-1} and l_{n-2} meet L in y_1. Any point between x_{n-2} and x_{n-1} will not be visible to any other endpoint apart from u_{n-1} and l_{n-1}. We now add a segment s_n to the arrangement having endpoints u_n and l_n such that u_n is between x_i and x_{i+1} where $0 \leq i \leq n-2$ and l_n is between y_1 and y_0. Clearly the point l_n is visible only to u_{n-1} and l_{n-1}. It is not difficult to see that for some choice of u_n we can add $i+1$ visibility edges to $E(CVG)_{n-1}$. Therefore $|E(CVG)_n| = |E(CVG)_{n-1}| + 1 + i$ where $0 \leq i \leq n-2$. Hence it follows that $|E(CVG)_n|$ can range between $n-2+1$ (i.e. $n-1$) to $\binom{n-1}{2} + n - 2 + 1$ (i.e. $\binom{n}{2}$). □

3. Classes of Graphs Representable as CVG

We now consider the classes of graphs that can be (or can not be) represented by a CVG. Simple paths can be realized by vertical segments whose convex hull boundary contains all endpoints (Figure 3a). Similarly, any complete graph K_n can be realized by considering n completely hulled segments. Other configurations can be drawn to realize K_n (Figure 3b).

A *wheel* W_n is defined as $K_1 + C_{n-1}$ where the + indicates that we add an edge between the vertex in K_1 and each of the vertices in C_{n-1}. Figure 3c shows configurations of a set of segments which realizes W_{11} as a CVG. The configuration can be generalized for any n.

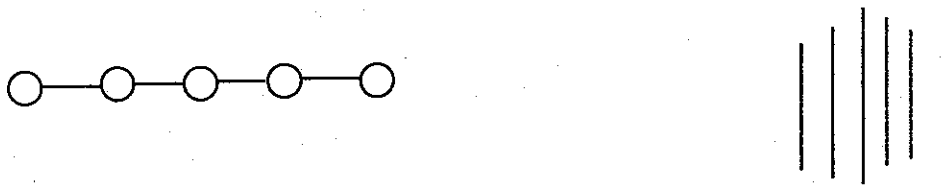
A *fan* F_n is defined as $K_1 + P_{n-1}$ where P_{n-1} stands for a path on $n-1$ vertices and the + indicates that we add an edge between the vertex in K_1 and each of the vertices in P_{n-1} (Figure 3d).

It is clear that if a configuration of segments contains a chord then the resulting CVG is non-Hamiltonian. A logical question might be whether a block in a CVG is Hamiltonian. Segment configuration examples exist where each block is non-Hamiltonian. Also the neighborhood of a

vertex in a CVG need not induce a Hamiltonian path.

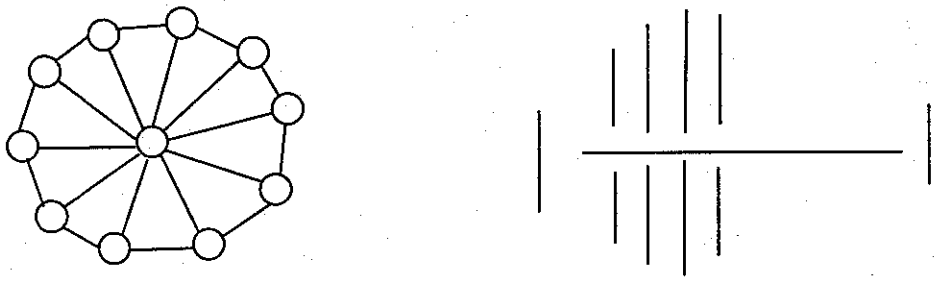
From Lemma 3 and Lemma 4, we can easily see that for $n > 3$, a cycle C_n cannot be realized as a CVG.

Lemma 4 implies that the only bipartite graph which is realizable as a CVG is a path. Any other CVG would contain K_3 (an odd cycle) making it non-bipartite.

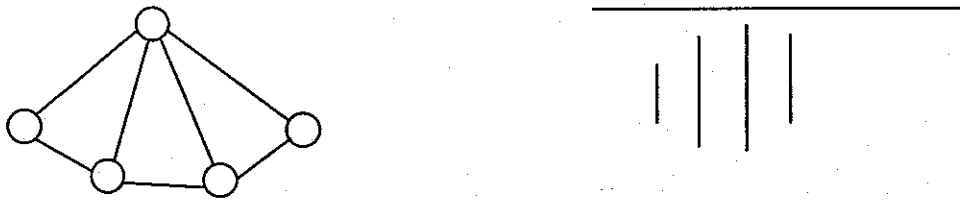


(a): Simple Path

(b): Complete Graph K_5



(c): Wheel Graph W_{11}



(d): Fan Graph F_5

Figure 3: Classes of Graphs Realizable as CVG

4. Concluding Remarks

We establish some properties of contracted visibility graphs (CVG) of disjoint line segments. There are several issues that need to be considered. For example, it would be interesting to determine the lower bound for the number of vertices in a block. Preliminary analysis shows that any block must have at least $2n - 3$ edges. However, we have not been able to establish this bound formally. We therefore state it as a conjecture.

Conjecture 1: The number of edges in a block in a CVG containing n vertices is greater than or equal to $2n - 3$.

One of the issues worth exploring would be to characterize the classes of graphs that can be represented by unit disjoint line segments.

Pocchiola and Vegter [PV94] define a weak visibility graph of the disjoint set of convex obstacles as one whose nodes are the obstacles and whose edges are pairs of obstacles such that there is a free line segment (bitangents) with endpoints lying on the obstacles. If the convex obstacles are replaced by straight line segments, then it is not difficult to observe that the bitangents are visibility edges between the endpoints of a pair of segments. From the definition of a weak visibility graph as introduced in [PV94] it is clear that what the authors call a weak visibility graph translates to a *CVG* when the obstacles are line segments.

We can introduce another class of visibility graph called *condensed visibility graph* (DVG) of disjoint line segments. In a DVG, two line segments are visible if all of their four endpoints are mutually visible. It is clear that DVG can be a proper subset of CVG but not the vice-versa. We have established some properties of DVG and these results will be reported in the future.

References

[BE*94] J. Bagga, J. Emert, M. Mcgrew, and W. Toll, "On the Sizes of Some Classes of Visibility Graphs", *Congressus Numerantium*, 104, (1994), pp. 25-31.

[CH91] D. Campbell and J. Higgins, "Minimal Visibility Graphs", *Information Processing Letters*, Vol. 37, (1991), pp. 49-53.

[LM*87] F. Luccio, S. Mazzone, and C. K. Wong, "A Note on Visibility Graphs", *Discrete Mathematics*, (1987), pp. 209-219.

[OS97] J. O'Rourke and Ileana Streinu, "The Vertex Edge Visibility Graph of Polygons", To appear in *Computational Geometry: Theory and Applications*, 1997.

[PV94] M. Pocchiola and G. Vegter, "Minimal Tangent Visibility Graphs", *Proceedings of the Sixth Canadian Conference in Computational Geometry*, 1994, pp. 24-29.

[R86] D. Rappaport, "The Complexity of Computing Simple Circuits in the Plane", PhD Thesis, McGill University, 1986.

[R92] J. Rippel, "Segment Visibility Graphs", Bachelor of Arts Thesis, Smith College, 1992.

[S96] T. Shermer, "On Rectangle Visibility Graphs", *Proceedings of the Eighth Canadian Conference on Computational Geometry, 1996*, pp. 234-239.

[SE87] X. Shen and H. Edelsbrunner, "A Tight Lower Bound on the Size of the Visibility Graphs", *Information Processing Letters*, Vol. 26, (1987/88), pp. 61-64.

[W89] S. Wismath, "Bar-Representable Visibility Graphs and a Related Network Flow Problem", Ph.D. Thesis, Department of Computer Science, University of British Columbia, 1989.

Geometric matching problem of disjoint compact convex sets by line segments

KIYOSHI HOSONO*

Department of Mathematics, Tokai University,
3-20-1 Orido, Shimizu, Shizuoka, 424 Japan

KATSUMI MATSUDA

Department of Mathematics, Tokai University,
316 Nishino, Numazu, Shizuoka, 410-03 Japan

April 5, 1997

1 Introduction

It is well known that any finite set of even points in the n-dimensional Euclidean space $\mathbf{R}^n$ admits a perfect matching by line segments. What is a suitable generalization of this fact from points to disjoint compact sets ? The case of disjoint line segments in the plane is treated in [1]. In this article, we survey some new results discussed in [2] and [3].

Definition 1.1. Let V be a disjoint family of compact sets in the n-dimensional Euclidean space $\mathbf{R}^n$, denoted by $\{C_a | \ a \in A\}$, and let L be a set of line segments in $\mathbf{R}^n$. The pair $F = (V, L)$ is said to be a *CL-figure* in $\mathbf{R}^n$, if V and L satisfy the following conditions:

1) Each endpoint of any line segment of L is on the boundary of a compact set C_a $(a \in A)$.

2) Any line segment of L has no common points with other line segments of L except possibly at common endpoints.

3) Any line segment of L has no common point with any C_a $(a \in A)$ except for its two endpoints.

If we regard the elements of V and the elements of L as the vertices and the edges, respectively, maintaining the incidence relation between V and L, then we can obtain a graph called the *skeleton* of the CL-figure $G = (V, L)$.

From now on, we will use the graph-theoretic terms in the skeleton of G to CL-figure G itself.

If V is a finite disjoint family, we call V an *even disjoint family* or an *odd disjoint family* if Card(V) is even or odd, respectively.

* *e-mail address*: hosono@scc.u-tokai.ac.jp

Definition 1.2. For a disjoint family V of compact sets, we define a *CL-matching of* V by a CL-figure $M = (V, L)$, if M satisfies the condition that any two elements of L cannot be adjacent to each other. We define $H(V)$ by $H(V) = max\{Card(L)|\ (V, L)\ is\ a\ CL\text{-}matching.\}$ and $h(m)$ by $h(m) = min\{H(V)|\ Card(V) = m\}$.

We call the set of the elements of V which are the ends of some $l \in L$ the *saturated ends set* $V(M)$. If $V(M) = V$, M is called a *CL-perfect matching of* V.

Furthermore, for an odd disjoint family V of compact sets and for an element E of V, we define a *CL-perfect matching (of V) with the residue* E by a CL-matching (V, L) of V such that $(V \setminus \{E\}, L)$ is a CL-perfect matching.

Definition 1.3. Let X^* denote the dual space of a linear space X. For a compact set C of $\mathbf{R}^n$ and for a non-trivial linear function $f \in (\mathbf{R}^n)^*$, let $m(C, f) = min\{f(x)|\ x \in C\}$ and $M(C, f) = max\{f(x)|\ x \in C\}$. $m(C, f)$ is called the *f-supporting value of* C. Define $d(C, f)$ by $d(C, f) = M(C, f) - m(C, f)$; $d(C, f)$ is called the *f-width of* C. C is said to be *f-connected* if $f(C) = [m(C, f), M(C, f)]$.

If $d(C_1, f) = d(C_2, f)$ for two compact sets C_1,C_2 of $\mathbf{R}^n$ and $f \in (\mathbf{R}^n)^*$, then C_1 and C_2 are said to have *f-equal width*. If V is a family of compact sets in $\mathbf{R}^n$, $V = \{C_a|\ a \in A\}$, and if f is an element of $(\mathbf{R}^n)^*$, then V is said to have *f-equal width* if C_a and C_b have f-equal width for any $a, b \in A$.

Definition 1.4. Let T be a set of points in $\mathbf{R}^n$. T is said to be *weakly 2-general* if T satisfies the following condition (C):

(C); For any point $P \in T$, there exists another point $Q \in T$ such that the line PQ does not contain any other point of T.

Let C be a compact convex set in $\mathbf{R}^n$. Let V be a family of translations of C in $\mathbf{R}^n$, denoted by $\{C + \overrightarrow{OP_i}|\ i = 1, 2, ..., n\}$. Then we call the set $\{P_i|\ i = 1, 2, ..., n\}$ the *translation set of* V, denoted by *Trans(V)*. V is said to be *weakly 2-general* if Trans(V) is weakly 2-general.

2 Main Theorems

We obtain the following results by the method in Section 3.

Theorem 2.1.

(1) There exists a CL-perfect matching for any even disjoint family of congruent discs in the Euclidean plane $\mathbf{R}^2$.

(2) For any odd disjoint weakly 2-general family of congruent discs in $\mathbf{R}^2$ and for any disc $E \in V$, V has a CL-perfect matching with the residue E.

Theorem 2.2.

Let f be a non-trivial linear function in the n-dimensional Euclidean space $\mathbf{R}^n$ $(n \geq 2)$ and V an even disjoint family of line segments in $\mathbf{R}^n$. If V has f-equal width where the f-supporting value of any element of V is distinct from others, then there exists a CL-perfect matching of V.

Let $e_i = (0, \cdots, 0, \overset{(i\,th)}{1}, 0, \cdots, 0)$ in $\mathbf{R}^n$ $(i = 1, 2, \ldots, n)$ and $\{f_i|\ 1 \leq i \leq n\}$ be the dual basis of $\{e_i|\ 1 \leq i \leq n\}$ in $(\mathbf{R}^n)^*$. Moreover, let D be a compact convex set in $\mathbf{R}^n$. A compact set E in $\mathbf{R}^{n+1}$ is said to be a *D-cylinder*, if $f_{n+1}^{-1}(\{c\}) \cap E$ is a translation of $D \times \{0\}$ in $\mathbf{R}^{n+1}$, for any $c \in \mathbf{R}$ such that $f_{n+1}^{-1}(\{c\}) \cap E \neq \emptyset$.

Theorem 2.3.
Let D be a disc in $\mathbf{R}^2$. Then there exists a CL-perfect matching for any even disjoint family V of D-cylinders in $\mathbf{R}^3$ which satisfies the following conditions:
1) V has f_3-equal width.
2) The f_3-supporting value of any element of V is distinct from others.
3) For sufficiently many $c \in \mathbf{R}$, if the set $\{E \cap f^{-1}(\{c\}) | \ E \in V \ and \ E \cap f^{-1}(\{c\}) \neq \emptyset\}$ is not empty, then it is weakly 2-general.

Theorem 2.4.
Let V be any disjoint family of $2n$ compact convex sets in $\mathbf{R}^2$. If V has f-equal width for some direction f, then it holds that $\lceil \frac{2n}{3} \rceil \leq h(2n) \leq \lfloor \frac{4n}{5} \rfloor$.

Theorem 2.5.
If V is a disjoint family of $2n$ translations of a convex body in the general position in $\mathbf{R}^2$, then it holds that $h(2n) \geq \lfloor \frac{4n}{5} \rfloor$.

3 Local Matching Principle

Let V be a finite disjoint family of compact sets in $\mathbf{R}^n$. If V satisfies the following condition for a linear function $f \in (\mathbf{R}^n)^*$ and a real number $c \in \mathbf{R}$, we say that V satisfies the *Local Matching Principle*(in short, *L.M.P.*) at the f-value c.

Condition: Consider any CL-matching $M = (V, L)$ of V such that
$V(M) \subseteq \{C \in V | \ C \subset_{\neq} f^{-1}((-\infty, c))\}$. Define V_c by $V_c = \{C \in V | \ C \cap f^{-1}(\{c\}) \neq \emptyset\}$. Then either of the following two conditions (L.M.P.I) or (L.M.P.II) holds:
(L.M.P.I)*$Card(V_c)$ is odd*;
There exists C_1 whose f-supporting value is minimum among the elements of $V \setminus (V(M) \cup V_c)$. Then there exists a set L_c of line segments in $\mathbf{R}^n$ such that the pair $(V_c \cup \{C_1\}, L_c)$ is a CL-perfect matching of $V_c \cup \{C_1\}$ and that the pair $(V, L \cup L_c)$ is a CL-matching of V.
(L.M.P.II)*$Card(V_c)$ is even*;
There exists a set L_c of line segments in $\mathbf{R}^n$ such that the pair (V_c, L_c) is a CL-perfect matching of V_c and that the pair$(V, L \cup L_c)$ is a CL-matching of V.

We state the algorithm by which we construct the "global" perfect matching by patching the "local" matchings. The following Main Lemma enables us to prove Thm.2.1, Thm.2.2 and Thm.2.3. Thm.2.4 and Thm.2.5 can be proved by the modified version of this lemma.

Main Lemma (Algorithm for the construction of a CL-perfect matching)
Let V be an even disjoint family of f-connected compact sets in $\mathbf{R}^n$ which has f-equal width for a linear function f. If V satisfies the following conditions, then there exists a CL-perfect matching of V.
1) The f-supporting value of any element of V is distinct from others.
2) V satisfies the Local Matching Principle at the f-value c for any $c \in \mathbf{R}$ except for a finite set of real numbers.
Proof.
1°) We shall construct the ascending chain $M_1, M_2, \ldots$ of CL-matchings of V. Since

$L_1 \subset_{\neq} L_2 \subset_{\neq} \cdots$ holds for $M_n = (V, L_n)$, then $V(M_1) \subset_{\neq} V(M_2) \subset_{\neq} \cdots$ also holds for the saturated ends sets. Thus $V = V(M_n)$ holds for some natural number $n \in \mathbf{N}$ by the finiteness of V. Therefore a CL-perfect matching of V results.

2°) Consider $V = \{C_1, C_2, \ldots, C_{2n}\}$ such that $c_1 < c_2 < \cdots < c_{2n}$ holds for $c_i = m(C_i, f)$ $(i = 1, 2, \ldots, 2n)$ by condition 1°). Since V has f-equal width, we simply denote the common value $d(C_i, f)$ for any i by d. Note that $f(C_i) = [c_i, c_i + d]$ by the f-connectivities.

3°) The construction of the CL-matching M_1 of V:

Since only C_1 and C_2 among the elements of V intersects the closed convex region $f^{-1}([c_1, c_2])$, we can join C_1 to C_2 by some line segment l_1 in this region. Then $(V, \{l_1\})$ is a CL-matching of V. Let $L_1 = \{l_1\}, M_1 = (V, L_1)$. Then choose some positive number $\varepsilon_1 > 0$ such that $c_2 < c_2 + \varepsilon_1 < min\{c_2 + d, c_3\}$ hold and the value $c_2 + \varepsilon_1 + md$ is never equal to c_3 for any $m \in N$.

4°) The construction of M_{n+1} by the CL-matching M_n of V:

For the constructed CL-matching $M_n = (V, L_n)$ of V and sequence of positive numbers $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$, let $V(M_n) = \{C_1, C_2, \ldots, C_{2k}\}$ and $L_n = \{l_1, l_2, \ldots, l_k\}$. Especially, suppose that the value $c_{2k} + \varepsilon_n + md$ is never equal to c_{2k+1} for any $m \in N$.

We can now construct a CL-matching M_{n+1} of V. There exists a unique natural number $m(n)$ by the choice of ε_n such that $c_{2k+1} < c_{2k} + \varepsilon_n + m(n)d < c_{2k+1} + d$.

Let $e(n) = c_{2k} + \varepsilon_n + m(n)d$. Then all the elements of $V(M_n)$ are included in the open half-space $f^{-1}((-\infty, e(n)))$, and $C_{2k+1} \in V_n$ holds for $V_n = \{C \in V |\ C \cap f^{-1}(\{e(n)\}) \neq \emptyset\}$. Therefore either of the following two conditions (I) or (II) holds since M_n satisfies the Local Matching Principle at the f-value e(n).

(I)*$Card(V_n)$ is odd*;

Denote $V_n = \{C_{2k+1}, C_{2k+2}, \ldots, C_{2k+2l-1}\}$ and let L'_n be some set of line segments in $\mathbf{R}^n$. Then by (L.M.P.I) there exists $(V_n \cup \{C_{2k+2l}\}, L'_n)$ of a CL-perfect matching of $V_n \cup \{C_{2k+2l}\}$ and $(V, L_n \cup L'_n)$ of a CL-matching of V. Next, since we can choose some positive number $\varepsilon_{n+1} > 0$ such that $\varepsilon_{n+1} < d$ and the value $c_{2k+2l} + \varepsilon_{n+1} + md$ is never equal to $c_{2k+2l+1}$ for any $m \in N$, we can construct $M_{n+1} = (V, L_{n+1})$ for $L_{n+1} = L_n \cup L'_n$.

(II)*$Card(V_n)$ is even*;

Denote $V_n = \{C_{2k+1}, C_{2k+2}, \ldots, C_{2k+2l}\}$ and let L'_n be some set of line segments in $\mathbf{R}^n$. Then by (L.M.P.II) there exists (V_n, L'_n) of a CL-perfect matching of V_n and $(V, L_n \cup L'_n)$ of a CL-matching of V. Since we can choose some positive number $\varepsilon_{n+1} > 0$ such that $\varepsilon_{n+1} < d$ and the value $c_{2k+2l} + \varepsilon_{n+1} + md$ is never equal to $c_{2k+2l+1}$ for any $m \in \mathbf{N}$, we can construct $M_{n+1} = (V, L_{n+1})$ for $L_{n+1} = L_n \cup L'_n$. □

4 Conjectures

We propose two conjectures selected in [2] and [3].

Conjecture 4.1. Let V be any disjoint family of $2n$ compact convex sets in $\mathbf{R}^2$. If V has f-equal width for some direction f, then it holds that $h(2n) = \lfloor \frac{4n}{5} \rfloor$.

Let V be a family of compact convex sets in $\mathbf{R}^n$. The family V is said to have *equal width*, if V has f-equal width for any $f \in (\mathbf{R}^n)^*$.

Conjecture 4.2. Let V be an even disjoint family of compact convex sets in $\mathbf{R}^n$. If V has equal width, then there exists a CL-perfect matching of V.

In particular, we are interested in the family of congruent balls in $\mathbf{R}^n$.

Acknowledgement
The author would like to thank Professor David Rappaport for his valuable suggestions.

References

[1] K.Hosono,M.Urabe and M.Watanabe,Topics on line segments and polygons,*Discrete Math.* 151(1996)99-104.

[2] K.Hosono and K.Matsuda,On the perfect matching of disjoint convex sets by line segments,*submitted.*

[3] K.Hosono,On an estimate of the size of the maximum matching for a disjoint family of compact convex sets in the plane,*submitted.*

Handling rotations in the placement of curved convex polygons.

François Rebufat
LIP6
Université Pierre et Marie Curie
4, Place Jussieu
75252 Paris Cedex 05
e-mail : rebufat@posso.ibp.fr
Fax : +33 1 44 27 40 42

April 4, 1997

1 Introduction

Curved polygons are compact subsets of $\boldsymbol{R}^2$ whose boundaries are build of lines segments and arcs of parabolas.

The problem addressed in this paper is to compute the translation set that sends a convex curved polygon in intersection with an other. It is a fondamental step for handling the computation of the space configuration for a mobil curved robot in a curved environment.

Basing our proposal on the Minkowski sum, we represent the space of contact-free configurations for A in B as the complementary set of $Sym(A) \oplus B$ where $Sym(A)$ is the symmetric of A with respect to the origin.

Early works have treated the case of "linear" polygons whose boundaries are only made of line segments [AB] or arcs of circles [JPL] and the case of curved polygons restricted to translations [JJR].

We propose a generalization of both cases allowing rotation/translation to convex curved polygons. More precisely, we present a theorical study as a fondation for an efficient algorithm to compute the application $\mathcal{F}$. Give two convex curved polygons A and B, $\mathcal{F}(A, B)$ is $(R_\theta(A) \oplus B, \Theta)$ the subset of $\boldsymbol{R}^2 \times \mathcal{S}^1$, where R_θ is the rotation with angle Θ.

In section 2 we expose briefly the computation of the Minkowski sum of two curved polygons. In section 3 we present a general algorithm for the placement of curved convex polygons in translation/rotation.

2 Sum in translation

We represent curved polygons by their boundaries i.e. by line segments and arcs of parabolas. We adopt a parametric representation of parabolas (in t or u) upon the interval $[0, 1]$ for simplicity.

Let $\gamma(t) = (X(t),\ Y(t))$ be a parabola, we define its derivative by $\gamma'(t) = (X'(t),\ Y'(t))$. While evaluating the derived curve for a value t_1 of t, we obtain the tangent vector to the curve γ at

the point $\gamma(t_1)$. We formalize the notion of tangent through the definition of tangential angular value.

Definition 2.1 *Let A be a curved polygon, we call tangential angular value(s) associated to $a \in \delta A$, noted $vat(a)$, the value(s) of angle(s) defined as follows :*
- if a is a vertex of A defined with the curves $A_1(t)$ and $A_2(t)$, $vat(a)$ is the set of values of the angles of the half-lines included between the half-tangents in a, $A_1'(1)$ and $A_2'(0)$.
- if $a \in A(t)$ then $vat(a)$ is the value of the angle of the half-tangent in a.

A *vat* is a real interval eventually reduced to a point. A list of *vat* can be sorted by using a real ordering relation. This allows us to orient a curved polygon according to its *vat* (setting a starting point). Arbitrary, we chose the counterclockwise orientation.

Now, we are able to discribe the construction of the Minkowski sum of two convex curved polygons. It is based on a criterion depending on the *vats* associated to the polygons.
The following proposition gives the criterion that will allow us to construct $\delta(A \oplus B)$.

Proposition 2.1 *Let A and B two convex curved polygons and let $a \in \delta A$ and $b \in \delta B$,*
$(a+b) \in \delta(A \oplus B) \iff vat(a) \cap vat(b) \neq \emptyset$.

Differents demonstrations can be found in [JJR], [JPL], [GSR], [FR].
To compute $\delta(A \oplus B)$ we proceed as follows : the *vats* of δA and δB are computed on each vertex and are sorted on the trigonometric circle. $\delta(A \oplus B)$ is built step by step while turning around following the growth of their tangential directions.
We note $Ai(t)$ an edge of A and a_i a vertex (respectively $B_i(u)$, b_i, for B). Curved that build the boundary of $\delta(A \oplus B)$ can be of three types :

1. The sum of one vertex of A and an arc of B :
$$B_i(u) + a_j$$
On the arc $B_i(u)$, $vat(B_i(u)) \subset vat(a_j)$.

2. The sum of one vertex of B and an arc of A :
$$A_i(t) + b_j$$
On the segment $A_i(t)$, $vat(A_i(t)) \subset vat(b_j)$.

3. The sum of an arc of A and an arc of B, such that *vats* are equal on these two arcs. To compute this, we need to express the relation ϕ that links parameters t and u such that $vat(A_i(t)) = vat(B_j(\phi(t)))$. To compute ϕ, we set to zero the determinant of the derivative matrix of curves components. ϕ is expressed as follow:
$$\phi(t) = \frac{\alpha t + \beta}{\delta t + \gamma},$$
where $\alpha, \beta, \gamma, \delta$ depend of the coefficients of curves. In [FR] we have proved that in the case of two parabolas the denominator of ϕ canot be equal to zero.

Following the *vats* on δA and δB such that tangential directions are equal gives a method to compute $\delta(A \oplus B)$. It shows which curve (or vertex) of A will be summed with which curve (or vertex) of B. The interval of variation for the parameter of each resulted curve is defined

during the process. Each time the configuration 3 appears, the relation ϕ is computed for the corresponding curves.

3 Minkowski sum with rotation.

Now the position of polygon A depends on a rotation parameter $\theta \in \Theta$ (typically $\Theta = [0, 2\pi[$) and will be noted $A(\theta)$. We extend the algorithm described above to this new hypothesis.
We note $R_\theta = (\rho_1(\theta), \rho_2(\theta))$ the rotation vector, classicaly, $(\sin(\theta), \cos(\theta))$.
The application of R_θ to a curve $\gamma(t)$ of δA, defines a family of curves $\gamma(t, \theta)$ called a parametric surface.

$$\gamma(t,\theta) = \begin{cases} \rho_1(\theta)X(t) - \rho_2(\theta)Y(t) \\ \rho_2(\theta)X(t) + \rho_1(\theta)Y(t) \end{cases}.$$

Decomposition of the rotation set Θ

Intuitively, to compute $A(\theta) \oplus B$ we proceed as follows. We split a continuous set in a finite number of subsets such that, each subset can be caracterized by one or more constructive properties. More precisely, we split the rotation set Θ in subintervals, such that on each interval $[\theta_i, \theta_{i+1}]$ the structure of the boundary of $A(\theta) \oplus B$ is preserved.

Definition 3.1 *Given two curved polygons A and B, we associate to $A \oplus B$ the ordered list of the labels of curves and vertices of δA and δB used to build the boundary. We call it the* list of structure *of $\delta(A(\theta) \oplus B)$.*

Such a list can be augmented by adding to each element the interval of variation of the parameter of the curve.
For the sum with rotation, the list of structure is modified when θ describes Θ. That is why, we are looking for a partition of Θ such that, on each interval I_i of this partition, the list of structure is kept unmodified.

Proposition 3.1 *Let $\mathcal{V}_1 = (\alpha_1, ..., \alpha_n)$ and $\mathcal{V}_2 = (\beta_1, ..., \beta_k)$ be the lists of vats for the vertices (two for one vertex) of A and B. for all $i = 1..n$ and all $j = 1..k$ the partition of Θ induced by the covering of the union of the intervals $[\alpha_i - \beta_j\ ,\ \alpha_{i+1} - \beta_j]$ defines a partition such that on each interval the list of structure is preserved.*

Proposition 3.2 *Let $\{I_1, .., I_n\}$ be a family of real intervals such that $\bigcup I_i = I$, the partition of I induced by the covering of $\bigcup I_i$ contains $2n - 1$ intervals and is computable in time $O(n \log n)$.*

The family $\{I_i\}_{i=1..n}$ contains at most $2n$ bounds. To compute the partition we sort these $2n$ bounds in $\mathbb{R}$. We show by induction that the number of intervals computed is at most $2n - 1$. Some intervals can be reduced to a point in the partition. They can be eliminated from the partition, the structure of the boundary of $\delta(A(\theta) \oplus B)$ being continuous.
The computation of $\delta(A(\theta) \oplus B)$ being decomposable on seperate intervals of Θ, we can compute exactly the surfaces of $\delta(A(\theta) \oplus B)$ on each interval.

Parametric surfaces

Lets us consider an interval $I_i =]\theta_i, \theta_{i+1}[$ with an associated list of structure $\mathcal{L}$. Three types of curves appear in $\mathcal{L}$:

1. the sum of a vertex of A and a curve of B ;

2. the sum of a vertex of B and a curve of A ;

3. the sum of a curve of A and a curve of B .

Construction of a curve of type (1)

Let a be the vertex of A and $\gamma_B(t)$ the curve of B. On the interval I_i, a draws an arc of circle. The resulting curve is then $\gamma_B + a(\theta)$. We have to describe how the interval of variation of the curve $\gamma_B(t) + a(\theta)$ varies for θ in I_i.
Let us set that for $\theta = 0$, $vat(a) = [\alpha, \beta]$. Upon I, $vat(a(\theta)) = [\alpha + \theta, \beta + \theta]$. The bounds (lower and upper) of the interval of variation for $\gamma_B(t) + a(\theta)$ are expressed with θ, considering that $\gamma_B(t)$ is defined upon $[0, 1]$:
- **Lower bound** : $max(r, 0)$, where r is solution of the equation

$$vat(\gamma_B(t)) = \alpha + \theta$$

- **Upper bound** : $min(r, 1)$, where r is solution of the equation

$$vat(\gamma_B(t)) = \beta + \theta$$

These equations are of degree 1 in t because they come from derivatives of the parabola $\gamma_B(t)$. In fact, $vat(\gamma_B(t)) = \alpha + \theta$ is rewritten in the form $\gamma'_{B_Y}(t)/\gamma'_{B_X}(t) = \alpha + \theta$ (if we take the slope of the tangent as representation for the vat). They only admit one solution.

Construction of a curve of type (2)

Let b be the vertex of B and $\gamma_A(\theta, t)$ the curve of A. The resulting curve is $\gamma_A(\theta, t) + b$. If we set $vat(b) = [\alpha, \beta]$ as previously the bounds of the interval of variation of parameter t are obtained as roots of equations depending on $vat(\gamma_A(\theta, t))$.
- **Lower bound** : $max(r, 0)$, where r is solution of the equation

$$vat(\gamma_A(\theta, t)) = \alpha, \text{ or } vat(\gamma_A(t)) = \alpha - \theta.$$

- **Upper bound** : $min(r, 0)$, where r is solution of the equation

$$vat(\gamma_A(\theta, t)) = \beta, \text{ or } vat(\gamma_A(t)) = \beta - \theta.$$

These equations are of degree 1, thus admitting only one solution.

Construction of a curve of type (3)

We have to sum $\gamma_B(u)$ and $\gamma_A(\theta, t)$. We proceed as we have done for the case where only translations are allowed ; we compute the relation $\phi(t, \theta)$ that links the parameters t and u as explained previously. Computing the determinant of the matrix of the derived coordinates of γ_A and γ_B (in t and u) gives the expression of $\phi(t, \theta)$.

$$\gamma'_A(\theta, t) = \begin{cases} (at + b)\rho_1(\theta) - (ct + d)\rho_2(\theta) \\ (ct + d)\rho_1(\theta) + (at + b)\rho_2(\theta) \end{cases}$$

$$\text{and} \qquad \gamma'_B(u) = \begin{cases} eu + f \\ gu + h \end{cases}$$

We can express $u = \phi(t, \theta)$ with the coefficients of $\gamma'_A(\theta, t)$ and $\gamma'_B(u)$. Noting $\gamma'_{Ax}(\theta, t)$ the *X-axis* coordinate of $\gamma'_A(\theta, t)$ and $\gamma'_{Ay}(\theta, t)$ its *Y-axis* coordinate,

$$u = \phi(t, \theta) = \frac{f\gamma'_{Ay}(\theta, t) - h\gamma'_{Ax}(\theta, t)}{g\gamma'_{Ax}(\theta, t) - e\gamma'_{Ay}(\theta, t)}$$

Summing $\gamma_A(t, \theta)$ and $\gamma_B(u)$ evaluated on $u = \phi(t, \theta)$ we obtain :

$$\gamma_{A+B}(\theta, t) = \begin{cases} \frac{1}{2}e\phi(t, \theta)^2 + f\phi(t, \theta) + c_1 + \gamma_{Ax}(t, \theta) \\ \frac{1}{2}g\phi(t, \theta)^2 + h\phi(t, \theta) + c_2 + \gamma_{Ay}(t, \theta) \end{cases} .$$

The solution is a surface described with rational functions of degree lower or equal to 4 in t and 3

in θ. The introduction of θ does not change the fact that the denominator of ϕ is always different from zero.
Different configurations lead to a construction of type 3. These configurations depend on the intervals of variation (on t and u) upon which curves are summed.
We will note $\gamma_{A_{|[t_1,t_2]}}$ (resp. $\gamma_{B_{|[u_1,u_2]}}$) the portion of γ_A (resp. γ_B) defined on the sub-interval $[t_1..t_2]$ (resp. $[u_1..u_2]$). We assume that originally, each arc of curve is defined on $[0,1]$. They are four possible configurations for $\gamma_A + \gamma_B$:

1. $\gamma_{A_{|[t_1,t_2]}} + \gamma_{B_{|[0,1]}}$,
   vats at the extremities of $\gamma_A(t,\theta)$ are totally included between *vats* at the extremities of $\gamma_B(u)$;
2. $\gamma_{A_{|[0,t_2]}} + \gamma_{B_{|[u_1,1]}}$,
   the two curves overlappe themeselves ;
3. $\gamma_{A_{|[t_1,1]}} + \gamma_{B_{|[0,u_2]}}$,
   this configuration is symetrical to the previous one ;
4. $\gamma_{A_{|[0,1]}} + \gamma_{B_{|[u_1,u_2]}}$,
   this configuration is similar to the first one.

The partition of Θ ensures that upon an interval I, $\gamma_A + \gamma_B$ corresponds to one of these configurations. It is easy to detect it, taking some value for θ included in I, for instance θ_M the middle of I. Comparing the *vats* at the extremities of $\gamma_B(u)$ and $\gamma_A(t,\theta_M)$ we identify the corresponding configuration (we could put this information in the list of structure).
The result of the computation of $\delta(A(\theta) \oplus B)$ on each interval of Θ is a family of parametric surfaces (in t or u), where the bounds of intervals of variation are expressed as functions of θ.

Combinatorial bound

For two convex curved polygons A and B having respectively n and m edges, computing their Minkowski sum takes $O(n+m)$ operations.
As we have seen previously, describing the partition of Θ is made in $O(nm \log nm)$ steps by sorting $O(nm)$ numbers.

Proposition 3.3 *The total complexity for computing the Minkowski sum of two convex curved polygons, one beeing allowed to move in rotation takes $O(nm \log nm)$ operations.*

Adjacency relations between surfaces on the same interval I_i of Θ (i.e. horizontal adjacency relation) are given by the list of structure.
Vertical adjacency relations are easely established since only one element of the list of structure is modified when we step from an interval I_i of the partition of Θ to the following interval I_{i+1}.
This algorithm gives a total description of the boundary of $A(\theta) \oplus B$ in $O(nm \log nm)$.

4 Conclusion

We have shown that problem of placement for two convex curved polygons in rotation/translation has same complexity as for convex linear ones [FA]: $O(nm \log nm)$. However, applications are more expensive in CPU time. Bigger data structures (curves vs. points) and more algebraic manipulations (computing the relation ϕ) increase significantly processing times. For the these reasons, implementation is more difficult to develop.
These results increase drastically for non-convex configurations because algebraic manipulations

become uneasy. We also need to compute all the triple-contact positions by solving in $\boldsymbol{R}$ non trivial polynomial systems.
Experiences led with the symbolic computation system AXIOM show that such applications are too expensive for real time robot motion planing. But applications could be developed for preprocessing configuration spaces or other needs like industrial cutting.

References

[JJR] J.J. Risler, *Placement of curved polygons,* AAECC -9, Lecture Notes in Computer Science 539 (1991), 368-383.

[PS] P. Schapira, *Operations on Constructible Functions,*
rapport DMI de Paris Nord, 88-2, serie mathématique, *1988.*

[AB] F. Avnaim et J.D. Boissonnat, *1989 Polygon placement under translation and rotation,*
Informatique Théorique et Applications (vol.23, n°1, 1989, p.5 à 28).

[JPL] J.P. Laumond, *1986 Obstacle growing in a nonpolygonal world,*
Information Processing Letters 25 (1987), 41-50.

[GSR] Guibas, Ranshaw, Stolfi, *A Kinetic framwork for computational geometry,*
proc. IEEE symp. on found. of Comp. Sci. (1983), 74-123.

[FA] F. Avnaim, *Placement et Deplacement de formes rigides ou articulées.*
Thèse de doctorat en sciences, Université de Franche-Compté, Juin 1989.

[FR] F. Rebufat, *Placement de Polygones Généralisés dans le Plan.*
P.h.D. thesis, Université Paris 6, April 1997.

Almost Optimal On-line Search in Unknown Streets

Evangelos Kranakis* Anthony Spatharis*

April 21, 1997

Abstract We consider the *on-line navigation problem* of a *tactile* robot searching for a target point g from a starting vertex s in an initially unknown *street*, which is a simple planar polygon (P, s, g) characterized by the property that the two oriented chains from s to g are mutually weakly visible.

We first present a deterministic competitive strategy of searching in unknown streets which achieves an almost optimal competitive ratio of $\frac{2+\sqrt{5}}{2\sqrt{2}}$ (≈ 1.498) in the L_2 *Euclidean* metric and significantly improves the previously best known competitive upper bound [12] of 1.73. Second, we easily modify our strategy into a slightly different on-line algorithm **HLS** (*High Level Strategy*) which minimizes the *clad* (*continues local absolute detour*) [8, 11] and has a better competitive factor of $\frac{3+\sqrt{5}}{2\sqrt{2}}$ (≈ 1.85) than the best known bound of 2.03 for this class of strategies. These greedy strategies have simple analyses with an optimal lower bound of $\sqrt{2}$ (> 1.41) on the competitive ratio for searches in unknown rectilinear streets.

1 Introduction

The problem of searching is fundamental to almost every area of computer science and it is a classical problem in computational geometry and robot motion planning [16]. Variants of searching problems have been studied, including searching for a specific recognizable object in a known or unknown geometric environment with or without obstacles [16].

We consider the navigation problem, in which a *tactile* (or *visual*) robot (i.e., a robot with an on-board vision system) has to find a path from a point s (*starting point*) to another point g (*goal*) in an unknown simple polygon without obstacles. The robot search is based only on the local information that it gathers through visual sensors and so it can be considered as an *on-line problem* [18]. Hence, we use the notion of *competitive analysis* of *on-line algorithms* [17] to measure the performance of this path planning problem. In other words, an on-line search strategy is defined to be *c-competitive* if the length of the path traveled by the robot is at most c times the optimal distance from s to the target point g. This constant c is called the *competitive ratio* (or *competitiveness*) of the strategy. Competitive strategies have many advantages and sometimes exist even for problems whose optimal solutions would be *NP-hard* [5].

Lumelsky and Stepanov [13] earlier studied a similar problem when a robot with a tactile sensor moves in an unknown environment of non-convex obstacles. They were the first to provide an upper bound to the length of the robot's path with respect to some parameters of the environment.

Papadimitriou and Yannakakis [15] have first considered the *competitive analysis* to measure the quality of on-line algorithms in motion planning theory. Several on-line problems have been studied for *searching, exploring* and *mapping* [1, 2, 7, 18] in a geometric environment using *visual* information.

Klein [8] first studied another navigation problem in a special polygon so-called a *street*. A simple planar polygon (P, s, g) with two distinguished vertices, s and g, is a *street* if and only if the two boundary oriented chains L (*left*) and R (*right*) from s to g are mutually weakly visible (i.e., each point of L can be seen from at least one point of R and vice versa). He described an on-line strategy to find a short path from s to g in a street, which achieved a competitive factor of $(1+\frac{3\pi}{2})$ (< 5.72) in the *Euclidean metric* L_2. This strategy has a lower bound of $\sqrt{2}$ (> 1.41) on the competitive factor to search in an initially unknown street.

Kleinberg [9] has considered a simple on-line algorithm for the same problem improving the competitive ratio to $2\sqrt{2}$ (< 2.83), although a tighter analysis [11] yields an upper bound of $2\sqrt{1+\frac{1}{\sqrt{2}}}$ (≈ 2.61).

*School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada. E-mail: {kranakis, spathari}@scs.carleton.ca. Part of this research was supported by NSERC (Natural Sciences and Engineering Research Council of Canada) grant.

He also proved that his strategy has an optimal $\sqrt{2}$-competitiveness for searching in *rectilinear* streets.

López-Ortiz and Schuierer [12] presented a on-line strategy, which has a better competitive ratio of $2\sqrt{1+(1+\frac{\pi}{4})^2}$ (≈ 2.05). Recently, they presented a *hybrid* strategy [11] which achieves a competitive ratio of 1.73 improving significantly the previous best known result.

Datta and Icking [3] defined a new, strictly larger class of simple polygons, so called *Generalized streets* (*G-streets*, for short) and presented an on-line strategy which achieves an optimal 9-competitive ratio (resp., $\sqrt{82}$-competitive) in *Manhattan metric* L_1 (resp., L_2) metric to search in an unknown rectilinear G-street. Moreover, López-Ortiz [10] has proposed a strategy with competitive ratio of 80 in L_1 metric to search in arbitrary oriented G-streets.

In addition, an even larger class of rectilinear simple polygons is given by the class of *HV-streets* with the property that every boundary point is mutually weakly visible from a point on a horizontal or vertical line segment connecting the two boundary oriented chains L and R from s and g. Recently, Datta [4] presented an optimal 14.5-competitive strategy in L_1 metric to search in rectilinear *HV-streets*.

In this paper we consider an efficient on-line navigation problem of a tactile robot in an unknown geometric environment. Particularly, we present a heuristic on-line algorithm which has a competitive ratio of $\frac{2+\sqrt{5}}{2\sqrt{2}}$ (≈ 1.498) in L_2 metric to visually search in unknown streets.

The remainder of this paper is organized in five sections. Section 2 contains the basic geometric concepts and important preliminary results. In Section 3, we describe our deterministic on-line strategy for searching in arbitrary unknown streets. Section 4 gives a very simple analysis to prove the competitive ratios of the proposed algorithm and its different versions. In section 5, we informally show that the above competitive factor appears to be the competitive lower bound for searching in arbitrary unknown streets. Finally, Section 6 summarizes our result and concludes with some open problems.

2 Preliminary Results

We begin with some definitions concerning general geometric concepts and we state the visibility properties of streets, before we describe the on-line strategy itself.

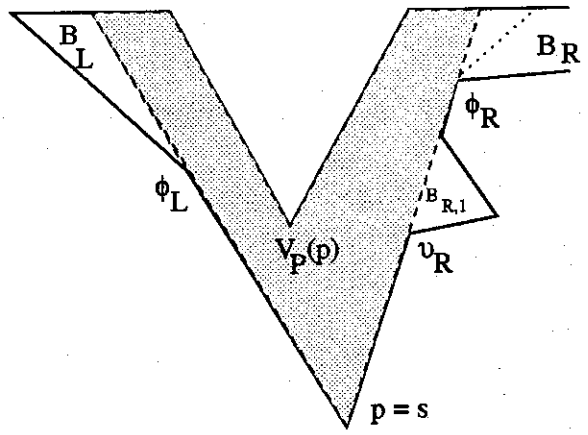


Figure 1: Visibility in a V-shaped Street

Let (P, s, g) be a street with a *starting* point s, and a *goal* point g on the boundary of polygon P, which is denoted by *bd(P)*. For simplicity, we assume that no three vertices of P are collinear and define the clockwise (resp., counterclockwise) polygonal chain from s to g to be the *left* (*right*) *chain* which is denoted by L (resp., R). The *visibility polygon* $V_P(p)$ of the polygon P from a point p in P is the set of $y \in P$ such that y is visible from p. A *window* w of $V_P(p)$ is an edge of $V_P(p)$ that does not belong to *bd(P)*. Clearly, a window w splits P into a number of sub-polygons P_1, P_2, ..., P_k one of which contains $V_P(p)$. Let us denote by P_w the union of the sub-polygons that do not contain $V_P(p)$. An *entrance point* (*sight point*) of a window w is the closer endpoint to p (see Figure 1).

We define a *bay* B of a window w to be the connected chain of *bd(P)* such that the robot has seen the endpoints of w but no other points of it. Now, let us assume that a window w has the orientation of the ray from p to the entrance point of w. We say that a window w is *left* (resp., *right*) *window* w_L (resp., w_R) if P_w is locally to the left (right) of w with respect to the given orientation of it. A *left* (*right*) *bay* B_L (B_R) is defined similarly.

We have the following lemma about the bays and entrance points of a window, which can be proved easily since it is similar to that in [9].

Lemma 2.1 *(i) Let $p \in$ bd(L) (resp., bd(L)) and let Ψ be the left (resp., right) (s, p)-boundary subchain of P. If the robot moves from s to p in P, it will have seen every point on Ψ. (ii) If v_L (resp., v_R) is a left (resp., right) entrance point of a window w_L (resp., w_R) of $V_P(p)$, then it belongs to the boundary chain L (resp., R).*

Now, let us denote by Γ the path that the robot has traveled from s to a point p in P. We say that a window w with respect to Γ is a *real window*, if it is also a window of $V_P(p)$. The *extended visibility polygon* $EV_P(p)$ of P at a point p in P consists of all the boundary points of P that have been seen so far (i.e., $EV_P(p) = \cup_{p\in\Gamma} V_P(p)$, which is also denoted by $V(p)$). Two windows are called L (resp., R) *-consecutive* if the left (right) polygonal chains of $V_P(p)$ between them does not contain a window different from them. Similarly, two real windows are called *consecutive* if there is no real window between them.

We say that a *left* (resp., *right*) *pharos* ϕ_L (resp., ϕ_R) of a window w_L (resp., w_R) is the entrance point of the leftmost (resp., rightmost) real left (resp., right) window w_L (resp., w_R) of p in $V_P(p)$. In other words, when $V_P(p)$ is traversed from p in clockwise order, the left (resp., right) entrance point of the window w_L (resp., w_R) which is encountered last (resp., first) is the left (resp., right) pharos ϕ_L (resp., ϕ_R) of that window (see Figure 2). Finally, the *visibility angle* of p is defined to be the angle at p between ϕ_L and ϕ_R (i.e., $\angle(\overline{\phi_R p}, \overline{p\phi_L})$ and is denoted by $v_\alpha(p)$.

We recall that a polygonal path from a point p_1 to another point p_2 is *monotone* if the x- and y-coordinates of its points never decrease along the direction of the straight-line path $\overline{p_1p_2}$). Also, let us denote by $d(p_1, p_2)$ the L_2 length of the shortest path $\mathcal{T}$ between two points p_1 and p_2 in P. We have the following lemma about real windows and pharos of $V_P(p)$.

Lemma 2.2 *(i) If w is a left (resp., right) window of $V_P(p)$ and bd(P) $\in$ L (resp., R), then w is not a real window. (ii) If g is contained in a bay B of a window w (left bay B_L or right bay B_R) and a point p belongs to the shortest path, then the (p, g)- path of $\mathcal{T}$ touches either left (right) pharos ϕ_L (resp., ϕ_R) of w. (iii) All windows that belong to L (resp., R) are L (resp., R) -consecutive in $V_P(p)$.*

Proof: (*i*) can be proved easily by contradiction, while the proofs of (*ii*) and (*iii*) are similar to those in [11, 12]. ■

Corollary 2.3 *(i) Real left (resp., right) windows are consecutive. (ii) If a window w_L (resp., w_R) is intersected by the robot's path, then all real left (right) windows are L (resp., R) -consecutive from w_R (w_L) in $V_P(p)$.*

3 The Competitive Strategy

In this section we describe our simple on-line strategy and its slightly different versions that consist of four cases which are based on the robot's extended visibility of a street.

Algorithm Street-CSS;

/* *Competitive Search Strategy* : This strategy finds a short path from a boundary starting point s to another point g in an initially unknown general street P and achieves a lower bound of $\frac{2+\sqrt{5}}{2\sqrt{2}}$ in L_2 metric on the ratio of the distance traveled by the robot to the length of a shortest path. */

Input : A street (P, s, g) and the robot is currently at the point s.

Output : The robot reaches the goal g.

begin ***Street-CSS***

$p := s$;

Determine EV_P(p), the pharos ϕ_L and / or ϕ_R;

/* it depends on their existence. */

while g is not visible in EV_P(p) **do**

Case 2. **If** the left (right) pharos ϕ_L (resp., ϕ_R) is defined, **then** the robot walks towards $p' := \phi_L$ (resp., $p' := \phi_R$); see Figure 2.

Case 3. **If** p, ϕ_L, ϕ_R are collinear, **then** the robot travels along their line to the closest pharos p' of ϕ_L and ϕ_R (see Figure 2).

Case 4. **If** none of the above two cases apply, **then** both pharos ϕ_L and ϕ_R are defined (see Figure 3).
If the visibility angle $v_\alpha(p) < \frac{\pi}{2}$ ($v_\alpha(p) \geq \frac{\pi}{2}$, respectively), **then** the robot chooses the positive direction of the y-axis such that ϕ_L is to the left and ϕ_R is to its right (resp., the positive direction on the line perpendicular to $\overline{\phi_L\phi_R}$). Next, it begins traveling in this direction *monotonically*, updating the extended visibility $EV_P(p)$ and the entrance points v_L, v_R, until it either sees the goal g or one of the following events occurs at some point **0** (i.e., the origin of the coordinate system):

1. One of the associated bays B_L (resp., B_R) with the entrance point v_L (resp., v_R) becomes completely visible.

2. **If** the current robot's position **r** is collinear to the entrance points v_L and v_R, **then** it moves to the closest entrance point p' of (v_L, v_R), which belongs to the shortest path $\mathcal{T}$.
3. **If** none of the above two events occur, **then** the robot travels along the vertical axis until a left (resp., right) entrance point v_L (resp., v_R) has the same y-coordinate as the robot and there is no other entrance point (except possibly its collinear entrance points on the horizontal axis) to the left (resp., right) side of the y-axis.

If event (1) happens first, **then** the robot walks directly to the opposite entrance point $p' := v_R$ (resp., v_L), thereby returning to the shortest path $\mathcal{T}$. **If** event (3) occurs, **then** the robot reaches the origin **0** and chooses the direction $\overline{0v_L}$ (resp., $\overline{0v_R}$) as the positive direction of x-axis. Now, it continues to move in the positive direction of the line $y = \frac{1}{2}x$ updating again the extended visibility $EV_P(p)$ and the entrance points v_L, v_R, until it sees the goal g or it reaches some target point **c** of the line $\overline{v_L v_R}$ where the event (1) always occurs. In this case, the robot moves directly to the entrance point p' $:= v_L$ (resp., v_R) which it has first seen from the origin **0**, thereby returning to the shortest path $\mathcal{T}$ again.

Set $p := p'$;

Compute $EV_P(p)$, the pharos ϕ_L and / or ϕ_R;

Updates $EV_P(p)$, the pharos ϕ_L and / or ϕ_R;

end; /* **while** */

Walk straight towards g; /* *Case 1* */

end.

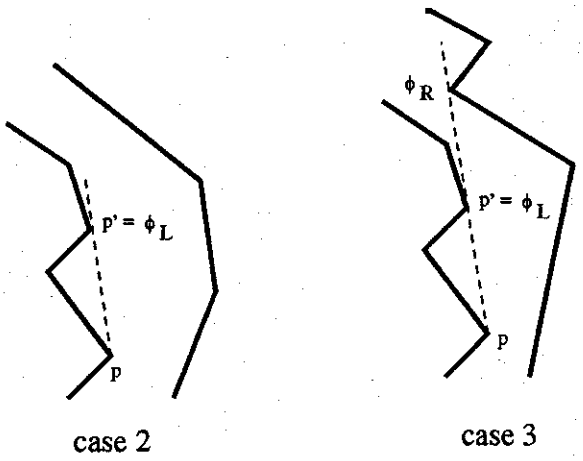


Figure 2: Robot's Movement in Cases 2 and 3

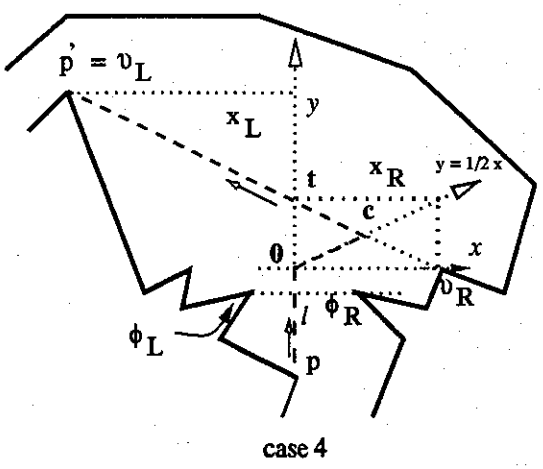


Figure 3: Robot's Movement in Case 4

4 Analysis of the Algorithm

The following lemma 4.1, which follows immediately from our algorithm **Street-CSS**, shows the correctness of this competitive strategy applying it repeatedly.

Lemma 4.1 *If one of the cases* $1-4$ *of the strategy* **Street-CSS** *holds at some current point* p *of the robot's path* Γ *and* $p \in \mathcal{T}$*, then the robot can always return to a later vertex* p' *of the shortest path* $\mathcal{T}$*.*

Next, the main inductive result of the competitive strategy is stated.

Theorem 4.2 *The deterministic on-line algorithm* **Street-CSS** *achieves a competitive ratio of* $\frac{2+\sqrt{5}}{2\sqrt{2}}$ *in* L_2 *metric to search in an unknown arbitrary street* P*.*

Proof: If only one of the cases $1-3$ applies (see, Figure 2) we easily get that the distance traveled by the robot from the entrance point p to p' is no more than $\sqrt{2}\cdot d(p,p')$ in L_2 metric, where $p, p' \in \mathcal{T}$.

Now, assume that case 4 occurs (see Figure 3). Let $l = |\overline{p0}|$ and $y_1 = |\overline{0t}|$. If the right entrance point v_R with coordinates $(x_L,\ y_R)$ has first seen, then the robot travels the distance $l + \sqrt{(x_R)^2 + (y_1)^2}$ from the entrance point p to the destination point $p' := v_R$ through the target point $\mathbf{c}$ in the line $\overline{v_L v_R}$, while we have $d(p,p') \geq \sqrt{l^2 + (x_R)^2}$. Thus, the worst-case competitive factor of our strategy is upper-bounded by

$$\sup_{l,x_R,y_L \in \mathcal{R}^+} \frac{l + \sqrt{(x_R)^2 + (\frac{y_L}{2})^2}}{\sqrt{l^2 + (x_R)^2}} \leq \frac{2+\sqrt{5}}{2\sqrt{2}}, \qquad (1)$$

where $\mathcal{R}^+$ denotes the set of non-negative real numbers.

If the destination point is $p' := v_L$, then the competitive ratio is bounded above by

$$\sup_{l,x_L,x_R,y_L \in \mathcal{R}^+} \frac{l + \sqrt{(x_L)^2 + (y_L - y_1)^2}}{\sqrt{(l+y_L)^2 + (x_L)^2}}$$

$$\leq \sup_{l,x_L,x_R,y_L \in \mathcal{R}^+} \frac{l + \sqrt{4(x_L)^2 + (y_L)^2}}{\sqrt{(l+y_L)^2 + (x_L)^2}} \leq \frac{1+\sqrt{5}}{\sqrt{5}}. \qquad (2)$$

The symmetric versions of the entrance and destination points are treated similarly, interchanging R with L properly etc. Also, we should point out that if the robot's motion is stopped by the boundary of P (e.g., see the V-shaped polygonal street of the Figure 1), then the preceding analysis and the competitive bounds of the above formulas (1) and (2) are not effected.

Clearly, this strategy creates a $(s,\ g)$-path that does not exceed $\frac{2+\sqrt{5}}{2\sqrt{2}}$ (≈ 1.498) times the shortest length $d(s,\ g)$ in L_2 metric. ■

Now, we can modify the case 4 of our strategy *Street-CSS* such that the robot continues moving on the positive direction of the y-axis until to reach the target point $\mathbf{t}$ of $\overline{\phi_L \phi_R}$ (see, Figure 3). This slightly different on-line algorithm **Street-HLS** (*High Level Strategy*) [8, 11] minimizes the *clad* (*continuous local absolute detour*).

Theorem 4.3 *The deterministic on-line algorithm* **Street-HLS** *has a competitive ratio of* $\frac{3+\sqrt{5}}{2\sqrt{2}}$ *(*≈ 1.85*) in* L_2 *metric for visual searching in an unknown general street.*

Proof: By a similar way to the proof of Theorem 4.2, the worst-case competitive ratio is upper-bounded by

$$\sup_{l,x_R,y_L \in \mathcal{R}^+} \frac{l + \frac{y_L}{2} + \sqrt{(x_R)^2 + (\frac{y_L}{2})^2}}{\sqrt{l^2 + (x_R)^2}} \leq \frac{3+\sqrt{5}}{2\sqrt{2}}. \qquad (3)$$

■

Furthermore, our proposed algorithm can be changed in such way that the robot continues traveling from the origin $\mathbf{0}$ to the pharos which has the same y-coordinate and from there to the opposite pharos. The new on-line algorithm **Street-LLS** (*Low Level Strategy*) achieves a competitive ratio of

$$\sup_{l,x_L,x_R,y_L \in \mathcal{R}^+} \frac{l + x_R + \sqrt{(x_L + x_R)^2 + (y_L)^2}}{\sqrt{(l+y_L)^2 + (x_L)^2}}$$

$$\leq \frac{2+\sqrt{5}}{\sqrt{5}} (\approx 1.89) \qquad (4)$$

in the L_2 metric and justifies the empirical bound reported by Klein [8].

Finally, we would like to point out that if the robot travels along the line $y = x$ (instead of $y = \frac{1}{2}x$), then our algorithm can achieve the competitive ratio of $1 + \frac{\sqrt{2}}{2}$ in L_2 metric. This factor is also better than the approximate competitive bound of 1.73 which has been proposed in[11].

Corollary 4.4 *If the street* P *is rectilinear, then our on-line strategies are optimal and have a competitive ratio of* $\sqrt{2}$ *in the* L_2 *metric.*

Proof: Since case 4 cannot occur when we apply our greedy algorithms in a rectilinear street. ■

5 A Competitive Lower Bound

In this section we informally show that our strategy **Street-CSS** appears to achieve a lower bound on the competitive ratio of $\frac{2+\sqrt{5}}{2\sqrt{2}}$ in L_2 metric.

Theorem 5.1 *The deterministic on-line algorithm* **Street-CCS** *has a lower bound on its competitive ratio of* $\frac{2+\sqrt{5}}{2\sqrt{2}}$ *in* L_2 *metric to search in arbitrary unknown streets.*

Proof: Let P denote the polygon shown in Figure 4, which is not a complete street since the goal has not yet been specified. The robot starts at the lower vertex s of an isosceles triangle $v_L p \phi_R$ and moves along the vertical y-axis to the middle point **0** of $\overline{v_L \phi_R}$. If the goal g is not visible yet and since no strategy can determine which one of the bays just around the pharos ϕ_L and ϕ_R contains the goal, the robot travels towards the point **c** on the interval $\overline{\phi_L \phi_R}$ following the positive direction of the line $y = \frac{1}{2}x$ (since the right pharos ϕ has the same y-coordinate as the robot). Then one of the associated bays, say B_R in this example, becomes visible at that target point **c**. Therefore, the robot continues moving straight towards the opposite pharos ϕ_L. But then the distance traveled by the robot from s to g is no more than $\frac{2+\sqrt{5}}{2\sqrt{2}} \cdot d(s, g)$ in L_2 metric using the *formula* (1) which has the minimum possible value by a simple numerical computation. ■

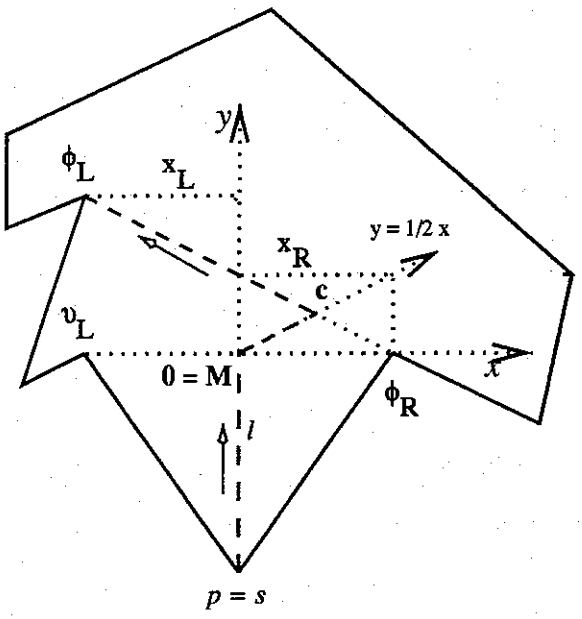


Figure 4: Establishing a Competetive Lower Bound

6 Conclusion

We have addressed the problem of efficient on-line navigation of a tactile robot in an unknown geometric environment and presented an on-line strategy that achieves a competitive lower bound of $\frac{2+\sqrt{5}}{2\sqrt{2}}$ (≈ 1.498) in the L_2 metric to search in unknown arbitrary streets.

The interesting *open question* remains if there exist more general natural classes of simple polygons larger than HV-streets [4, 10] which can be searched competitively. Furthermore, we want to find a strategy that achieves a better competitive ratio of 80 in L_1 metric for searching arbitrary G-streets.

A common research problem is to improve the competitive factor of 133 in L_2 metric for exploring an unknown simple polygon. Moreover, the main and most interesting problem in this field [5, 6] is the following: Is there a competitive algorithm for learning an unknown *rectilinear* polygon with an arbitrary number of *rectilinear* obstacles? Although, it is well known that the previous problem is not true for any number of arbitrary holes in general polygons, we could find some polygons for which the above problem is valid.

Generally, we would like to design optimal competitive strategies for these generalized problems and other applications in on-line navigation and motion planning theory, in which sophisticated optimization techniques could promise new and interesting results.

Acknowledgement We thank Anil Maheshwari for interesting discussions and useful comments on an earlier version of this paper.

References

[1] R. A. Baeza - Yates, J. C. Culberson, and G. J. E. Rawlins. Searching in a plane. *Information and Computation, Vol. 106*, pp. 234-252, 1993.

[2] R. A. Baeza - Y A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. In *Proc. 23rd ACM Symp. Theory of Computing*, pp. 494-504, 1991.

[3] R. A. Baeza - Y A. Datta and C. Icking. Competitive searching in a generalized street. In

10th ACM Computational Geometry, pp. 175-182, 1994.

[4] A. Datta, Ch. Hipke, and S. Schuierer. Competitive searching in polygons - beyond generalized streets. In *Proc, Sixth Annual International Symp. on Algorithms and Computation. L. N. C. S.*, pp. 32-41, 1995.

[5] X. Deng, T. Kameda, and C. M. Papadimitriou. How to learn an unknown environment I: The rectilinear case. *Technical Report CS-93-04, Department of Computer Science, York University, 1993.*

[6] F. Hoffmann, Ch. Icking, R. Klein and K. Kriegel. An efficient Competitive Strategy for Learning a Polygon. In *Proc. 8th ACM-SIAM Symp. on Discrete Algorithms*, 1997.

[7] B. Kalyanasundaram and K. K. Pruhs. A competitive analysis of nearest neighbor algorithms for searching unknown scenes. In *Proc. 9th Ann. Symp. on Theoretical Aspects of Computer Science*, pp. 147-157, 1992.

[8] R. Klein. Walking an unknown street bounded detour. In *Computational Geometry: Theory and Applications 1*, pp. 325-351, 1992.

[9] J. M. Kleinberg. On-line search in a simple polygon. In *Proc. of the 5th ACM-SIAM Symp. on Discrete Algorithms*, pp. 8-15, 1994.

[10] A. López-Ortiz and S. Schuierer. Generalized Streets Revisited. In European Symposium of Algorithms. *Springer Verlag, Lecture Notes in Computer Science, Vol. 1136*, pp. 546-558, 1996.

[11] A. López-Ortiz and S. Schuierer. Walking Streets Faster. In *Proc. of 5th Scandinavian Workshop on Algorithm Theory - SWAT '96. Springer Verlag, Lecture Notes in Computer Science, Vol. 1097*, pp. 335- 356, 1996.

[12] A. López-Ortiz and S. Schuierer. Going Home Through an Unknown Street. In Proc. of 4th Workshop on *Algorithms and Data structures. Springer Verlag, L.N.C.S., Vol. 955, pp. 135-146*, 1995.

[13] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. In *Algorithmica 2*, pp. 403-430, 1987.

[14] A. Mei and Y.1 Igarashi. An efficient strategy for robot navigation in unknown environment. In *Inform. Processing Letters 52*, pp. 5-56, 1994.

[15] P. Papadimitriou and M. Yannakakis. Shortest paths without a map. In *Int. Colloq. on Automata, Languages, and Programming (ICALP '89)*, pp. 60-620, 1989.

[16] J. T. Schwartz and M. Sharir. Algorithmic motion planning in robotics. *Handbook of Theoretical Computer Science. MIT Press*, pp. 391-430, 1991.

[17] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM, Vol. 28*, pp. 202-208, 1985.

[18] A. Spatharis. Efficient Dynamic and On-line Computation with Applications. *M. Sc. Thesis, School of Computer Science, Carleton University, Ottawa, Canada, 1995.*

An On-line Algorithm for Exploring an Unknown Polygonal Environment by a Point Robot

S.K. Ghosh
Computer Science Group
Tata Institute for Fundamental Research
Bombay 400005, INDIA
ghosh@tifrvax.tifr.res.in

J.W. Burdick
Dept. of Mechanical Engineering
California Institute of Technology
Pasadena, CA, USA 91125
jwb@robby.caltech.edu

Abstract

In this paper, we propose an on-line algorithm that can be used to explore an unknown polygonal environment by a point robot under the restriction that the robot computes visibility polygons at a discrete number of points on its path. We show that the exploration algorithm computes at most $r + 1 - h$ visibility polygons, where r is the total number of reflex vertices in the polygonal environment containing h holes. We also show that $r + 1 - h$ visibility polygons are sometime necessary to see the entire polygonal environment.

1 Introduction

In computational geometry literature, several algorithms, specifically on-line algorithms (see [2] and references contained therein), assume that the visibility region can be determined in a continuous fashion from each point on a path of a robot. While this assumption is reasonable in the case of a human "watchman," it is not practical in the robotic case for several reasons. First, autonomous robots can only carry a limited amount of on-board computing capability. At the current state of the art, computer vision algorithms that could compute visibility polygons are time consuming. The computing limitations dictate that it is not practically feasible to continuously compute the visibility polygon along the robot's trajectory. Second, for good visibility, the robot's camera will typically be mounted on a mast. Such devices vibrate during the robot's movement, and hence for good precision (which is required to compute an accurate visibility polygon) the camera must be stationary at each view. Therefore, it is only feasible to compute the visibility polygon at a discrete number of points. Hence we hereafter assume that the visibility polygon is determined at a discrete number of points. This notion of computing visibility from discrete points rather than in continuous fashion and the restriction of no apriori knowledge of the environment's geometry suggests a new framework in computational geometry for designing algorithms for motion planning.

Though the above discussion suggests that a robot can only compute visibility polygons at discrete points on its path, it is not clear whether total cost for a robotic exploration is dominated by the number of visibility polygons that it computes or the length of the path it travels. The computational geometry literature has typically assumed that the cost associated with a robot's physical movement dominates all other associated costs, and therefore minimizing the Euclidean length of the path of a robot is considered as the main criteria or the sole criteria for designing motion planning algorithms. The essential components that contribute to the total cost required for a robotic exploration can be analyzed as follows. Each move will have two associated costs. First, there is the time required to physically execute the move. If we crudely assume that the robot moves at a constant rate, r, during a move, the total time required for motion will be $r\ D$, where D is the total path length followed by the robot during the exploration. Second, in an

exploratory process where the robot has no apriori knowledge of the environment's geometry, each move must be planned immediately prior to the move so as to account for the most recently acquired geometric information. The robot will be stationary during this process, which we assume to take time t_M. Because straight line paths are the easiest to plan, and since any curvilinear path can be well approximated by straight line segments, we assume that each move consists of a straight segment. For the reasons outlined above, we assume that the robot is stationary during each sensing operation, which we assume takes time t_S. Let N_M and N_S be respectively the number of moves and the number of sensor operations that are required to complete the exploration of P. Hence the total cost of an exploration is equated to the total time, T, required to explore P:

$$T(P) = t_M\ N_M + t_S\ N_S + r\ D \tag{1}$$

Any practically relevant algorithm should endeavor to minimize this cost. Experience dictates that in a relatively cluttered environment, the actual time that is required to complete an exploration will be dominated almost entirely by the computation time needed to compute visibility polygons, since current computer vision algorithms consume significant time on modest computers. So, not only it is feasible to compute visibility polygon only from discrete points but also the overall cost of exploration is dominated by the cost for computing visibility polygons. Hence, in this paper we will use N_S as a measure of exploration efficiency

$$\text{Cost} \ = \ T(P) \ \simeq \ O(N_S)$$

and thus the goal is to develop an exploration algorithm for a robot that minimizes N_S.

2 An exploration algorithm for a point robot

In this section, we describe an on-line algorithm that a point robot can use to explore an unknown polygonal environment P and guarantee that entire free-space $\mathcal{F}$ of P has been seen by the robot. It may appear that in order to see the entire free-space, it is enough to see all vertices and edges of P. However, this is not the case, as shown by the example in Figure 1. In this figure, three views from p_1, p_2 and p_3 are sufficient to seen all vertices and edges of P but the shaded region of $\mathcal{F}$ cannot be seen from these views. This suggest that in order to see the entire free-space, the algorithm must ensure that all triangles in the triangulation of P have been explored.

The algorithm proceeds as follows. The robot starts at any initial location, p_1, where it determines the visibility polygon $V(P, p_1)$ from p_1. Using the visible vertices of P in $V(P, p_1)$, the robot triangulates as much of the $V(P, p_1)$ as possible. Let this triangulation be denoted T_1. The robot will then execute (in a manner to be described below) a forward move to p_2, where it will compute the next visibility polygon $V(P, p_2)$. The region common to $V(P, p_2)$ and T_1 is removed. The remaining free space in $V(P, p_2)$ is triangulated, and the total triangulation is updated.

To describe the algorithm in more detail, assume that the robot is beginning the i^{th} step of the exploration procedure, and is therefore situated at configuration p_i. Let T_{i-1} denote the cumulative triangulation that has been established prior to step i. Using its on-board sensors, the robot computes $V(P, p_i)$ and then removes the region common to $V(P, p_i)$ and T_{i-1} from $V(P, p_i)$. If this operation splits $V(P, p_i)$ in several disjoint parts, choose the part containing p_i as $V(P, p_i)$. As a result $V(P, p_i)$ contains a portion of $\mathcal{F}$ which is yet to be triangulated. It can be views as if $V(P, p_i)$ has been computed by treating T_{i-1} as an opaque region. From now on, whenever we refer to $V(P, p_i)$, it means that $V(P, p_i)$ is the connected unexplored region of $\mathcal{F}$, that is visible from p_i, and contains p_i.

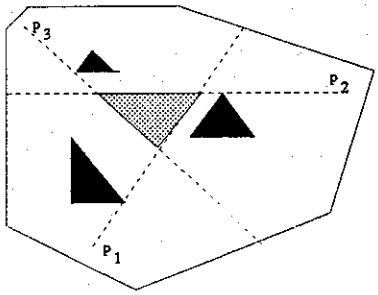


Figure 1: Three views are enough to see all vertices and edges but not the entire free-space.

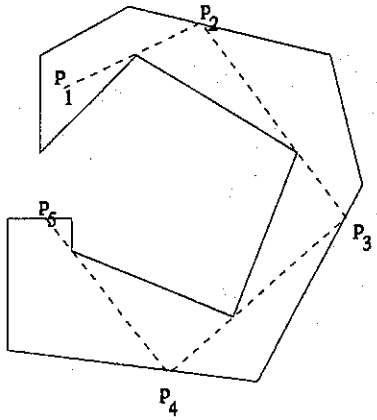


Figure 2: The spiral polygon can be explored in $r+1$ steps.

After computing $V(P,p_i)$, the interior of $V(P,p_i)$ is triangulated by connecting only the vertices of P that lie in $V(P,p_1)$. The vertices of the triangulation may also include vertices from previously triangulated regions. Let T_i' denote the triangulation of the newly viewed free-space in $V(P,p_i)$. Hence, at the end of step i, the total free-space triangulated is $T_i = T_{i-1} \cup T_i'$. In the following lemma, we show that T_i' is not empty.

Lemma 2.1 *There exists at least one triangle in* T_i'.

Proof: We know that p_i has been chosen in such a way that p_i does not lie inside T_{i-1} but lies inside $V(P,p_{i-1})$. Without loss of generality, we assume that p_i lies in the triangle uvw, where u and w are vertices of P, uv is a constructed edge of $V(P,p_{i-1})$ and uw is a boundary edge of T_{i-1}. Now, the boundary of $V(P,p_i)$ consists of the edge uw and a chain of polygonal and constructed edges connecting u and w. Since, this chain makes a turn of 180 degrees or more with respect to p_i, there exists a vertex u' of P on the chain such that $uu'w$ is a triangle inside $V(P,p_i)$. Hence, $uu'w$ belongs to T_i'. □

Corollary 2.1 *The point* p_i *lies inside* T_i' *and* $V(P,p_i)$ *removes at least one constructed edge of* $V(P,p_{i-1})$.

The above lemma suggests that every time a view is taken, at least one new triangle is explored for each forward move, and therefore the algorithm cannot compute more views than the number of triangles in the triangulation of P. As a by-product, at least one constructed edge is also removed during each forward move. Since a polygon with h holes and N total edges can be triangulated with $N+2h-2$ triangles, the algorithm can compute at most $N+2h-2$ views. This is generally not an efficient bound. For example, the robot can also make an effective exploration by tracing the boundary of P and taking a view at each vertex. This exploration would require only N views. However, a tighter bound can be proved as follows. Observe that every time a view is taken, a constructed edge uv is removed, where u is a reflex vertex of P as well as is a vertex of the free-space that has been triangulated so far. Since the algorithm considers the previously triangulated region as an opaque region for computing subsequent views, there cannot be another constructed edge

with u as vertex. So we can associate each view (except the first view) to a distinct reflex vertex of P. Therefore, the algorithm computes at most $r+1$ views, where r is the total number of reflex vertices of P, which is generally much less than N. We state this fact in the following lemma.

Lemma 2.2 *The exploration algorithm computes at most $r+1$ views.*

From the acquired view, the robot establishes a list of constructed edges, $(C_{i,1}, C_{i,2}, ..., C_{i,c_i})$, on the boundary the current visibility polygon. These edges will help to define the subsequent forward exploratory moves. Since we assume that the robot has a dead reckoning sensor, the coordinates of these constructed edges are assumed to be known. For each constructed edge, choose a point $z_{i,j}$ $(j = 1, \cdots, c_i)$ close to $C_{i,j}$ which will be used to view the unexplored territory that is associated with the j^{th} constructed edge. Observe that the choice of $z_{i,j}$ can play a major role in deciding the number of visibility polygons that are required to see the unexplored free-space. However, any specific strategy for choosing $z_{i,j}$ does not seem to perform well in all situations. So, it suffices to state that $z_{i,j}$ is not in T_i' and that the entire constructed edge $C_{i,j}$ can be seen from $z_{i,j}$.

Choose one of the $z_{i,j}$, say $z_{i,1}$ for example, to be the next viewing point, p_{i+1}, and recursively apply this procedure. Hence, the algorithm is a depth-first search of the unexplored free-space. When all of unexplored territory associated with $z_{i,1}$ has been explored, then choose another viewing point at level i, say $z_{i,2}$, and continue. Note that while choosing $z_{i,2}$ as the next viewing point, the corresponding constructed edge $C_{i,2}$ must lie partially or totally outside the free-space triangulated so far. Otherwise, it is considered that $C_{i,2}$ has been explored. A local exploratory procedure will terminate when:

1. the visibility polygon contains no constructed edges. In this case, the robot retraces its steps to the last unexplored constructed edge.
2. all constructed edges of a visibility polygon have been explored. The robot returns to the previous visibility polygon with unexplored constructed edges.

The algorithm terminates when all constructed edges of $V(P, p_1)$ have been explored recursively. In the following lemma we show that the algorithm has explored the entire free-space $\mathcal{F}$.

Lemma 2.3 *When all constructed edges of $V(P, p_1)$ have been explored recursively, the algorithm has explored the entire free-space $\mathcal{F}$.*

Proof: Assume on the contrary that all constructed edges of $V(P, p_1)$ have been explored recursively but there exists a point $z \in \mathcal{F}$ which has not been seen by the algorithm. Consider a path Q inside P connecting p_1 to z. If no such path exists, then z does not lie in the free-space $\mathcal{F}$ which contradicts the assumption that $z \in \mathcal{F}$. So we assume that such a path Q exists. Since p_1 lies in the triangulated region, starting from p_1, Q must intersect at least one boundary edge uv of the triangulation before reaching z. It means that there exists an unexplored constructed edge bounded by u or v, which contradicts the fact that all constructed edges of $V(P, p_1)$ have been explored recursively. Therefore, the entire path Q lies inside the triangulation of P. Hence, z has been seen by the algorithm. □

Let us now compare the performance of our algorithm with that of an optimal exploration algorithm. Consider a spiral polygon without any hole as shown in Figure 2. It can be seen from the figure that no algorithm, starting from p_1, can explore the entire spiral polygon in less than $r+1$ steps. On the

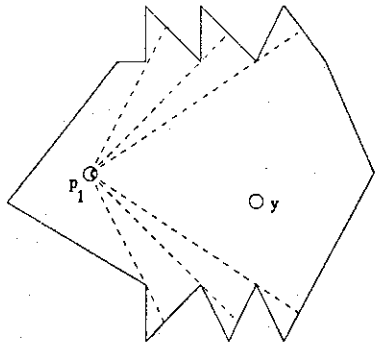


Figure 3: The star-shaped polygon can be explored in two steps.

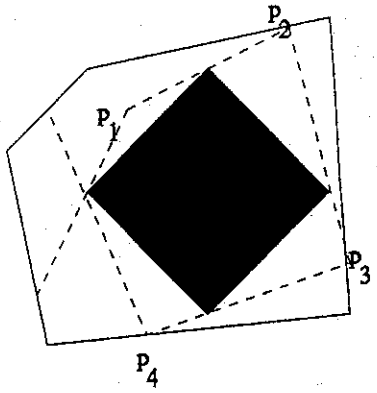


Figure 4: The polygon can be explored in r steps.

other hand, consider a star-shaped polygon without any hole as shown in Figure 3. In this figure, two views are enough to see the entire star-shaped polygon because robot takes the first view at the given starting position p_1 and then it takes the second view from the star-point y. On the other hand, our algorithm first takes a view from p_1 and then takes r views to remove all constructed edges of $V(P, p_1)$. This example shows that this is the worst performance of our algorithm (i.e., competitive ratio is $(r+1)/2$) with respect to the performance of an optimal exploration algorithm. Let us now consider polygons with only one hole. It can be seen from Figure 4 that our algorithm takes r steps to explore the entire polygon. In fact, this example shows that an optimal exploration algorithm also takes r steps to see the entire polygon. However, examples can easily be constructed to show that the performance of our algorithm may not be close to that of an optimal algorithm as more holes are added to the polygon. On the other hand, it seems that more and more holes are added to a polygon, our algorithm takes much less than $r+1$ steps to explore the entire free-space. In the following lemma, we prove that the tighter upper-bound of our algorithm is $r+1-h$, where h is the number of holes in the polygon.

Lemma 2.4 *The exploration algorithm computes at most $r+1-h$ views.*

Proof: Let $T_1(P)$ denoted the region of the free-space triangulated so far by the algorithm. Let ab and cd be the two boundary edges of $T_1(P)$ which are not polygonal edges. So there exists two constructed edges, say aa' and cc', which are yet to be explored. Suppose a point z_c, arbitrary close to cc', is chosen as the next viewing point. Since the exploration algorithm uses the depth-first search method, the algorithm will explore the entire free-space lying outside $T(P)$ that can be reached from z_c before a viewing point is chosen with respect to aa'. Let $T_2(P)$ be the triangulated region of the newly explored region starting from z_c. If there is a polygonal path from z_c to the edge ab inside $T_2(P)$, then no viewing point will be chosen with respect to aa' as ab no longer remains as a boundary edge of the triangulated region $T_1(P) \bigcup T_2(P)$. Moreover, while constructing $T_2(P)$ no constructed edge can be created with a as a vertex as any visibility polygon computed from z_c or any subsequent viewing point of z_c will treat $T_1(P)$ as an opaque region. Observe that such situation arises when the polygonal paths from z_c to some point of ab inside $T_1(P)$ and inside $T_2(P)$ enclose at least a hole. It implies that once the free-space around a hole is explored by the depth-first search method, a constructed edge generated earlier will be lying in the newly triangulated

region. Hence no viewing point will be chosen with respect to that constructed edge. Hence, the total number of views computed by the exploration algorithm can be at most $r+1-h$ even though $r+1$ constructed edges may be created during exploration. □

Lemma 2.4 provides the upper-bound on the computational complexity of the exploration. The correctness the exploration algorithm and the completeness of the exploration follow from Lemma 2.1 and Lemma 2.3 respectively. We summarize the result in the following theorem.

Theorem 2.1 *The exploration algorithm correctly explores the entire polygonal environment in at most $r+1-h$ steps.*

3 Concluding remarks

Suppose the given robot is a convex polygon C and the robot wants to explore the free-space of P. Let x be the point of C corresponding to the position of the sensor of the robot. If C knows P apriori, then the configuration space of P can be computed using Minkowski sum of C and P with x as the reference point and then use the algorithm mentioned in the previous section. Since C does not know P apriori, a different approach is required for this problem. We have designed an exploration algorithm which computes the configuration space (under translation) incrementally in $N+2h-2$ steps using a similar strategy as in the case of a point robot, and then discretizes the non-polygonal edges of the configuration space by a set of points so that the robot can take views from these points in order to see the free-space of P lying outside the configuration space. However, the visibility polygons computed from the chosen set of discrete points may not always guarantee to see all points of P. For details of the exploration algorithm, refer to [1].

Suppose p_1, p_2,...., p_k be the points of P such that an optimal exploration algorithm for a point robot has computed visibility polygons from these points. We know that $\bigcup_{i=1}^{k} V(P,p_i) = P$, $p_{i+1} \in V(P,p_i)$ and k is minimum. So, P can be guarded by placing stationary guards at p_1, p_2,...., p_k. So, the exploration problem for a point robot is the Art Gallery problem for stationary guards [3, 4] with additional constraints. Hence, our exploration algorithm for a point robot is an approximation algorithm for this variation of the Art Gallery problem which seems to be NP-hard.

4 References

1. S. K. Ghosh and J. W. Burdick, *Exploring an unknown polygonal environment with a sensor based strategy*, manuscript, 1997.

2. S. K. Ghosh and S. Saluja, *Optimal on-line algorithms for walking with minimum of turns in unknown streets*, Computational Geometry: Theory and Applications, vol. 6 (1997), 1-27.

3. J. O'Rourke, *Art gallery theorems and algorithms*, Oxford University Press, 1987.

4. T. Shermer, *Recent results in art galleries*, Proc. of the IEEE, 80(9):1384-1399, 1992.

Understanding Discrete Visibility and related Approximation Algorithms

S.K. Ghosh
Computer Science Group
Tata Institute for Fundamental Research
Bombay 400005, INDIA
ghosh@tifrvax.tifr.res.in

J.W. Burdick
Dept. of Mechanical Engineering
California Institute of Technology
Pasadena, CA, USA 91125
jwb@robby.caltech.edu

Abstract

In this paper, we consider the following question. Given a human 'watchman' tour that was computed with a continuous or weak visibility assumption, is it possible for a robot 'watchman' to realize the same visibility by using only a discrete number of views along the same path? In other words, given a continuous path, is there a discrete subset of points in that path from which one obtains the same effective visibility as would be obtained by the continuous or weak visibility assumption? We present 'off-line' and 'on-line' approximate algorithms to locate a set of discrete points on the path.

1 Introduction

In computational geometry literature, several algorithms, specifically on-line algorithms (see [3] and references contained therein) assume that the visibility region can be determined in a continuous fashion from each point on a path of a robot. While this assumption is reasonable in the case of a human "watchman," it is not practical in the robotic case. Ghosh and Burdick [1] have shown that it is only feasible to compute visibility polygons only from discrete points on a path of a robot and the overall cost is dominated by the cost for computing visibility polygons. Within this framework, we consider the following question. Let Q be a continuous path of a human watchman tour in P. It means that the entire free-space of P is visible from some point on the path Q, i.e., P is weakly visible from Q. Is there a set of discrete points $W = (w_1, w_2, ..., w_k)$ on Q such that $\bigcup_{i=1}^{k} \mathcal{V}(P, w_i) = P$? Here, we present approximation algorithms to locate a set W from a given path Q. Algorithms construct W consisting of both endpoints of each line segment of Q and some internal points on each line segment of Q (if required). So, it is enough to present the procedure for selecting a set of internal points for a line segment st of Q.

2 Limits to the discrete visibility approximations

Let us first consider the basic limits to the discrete visibility approximations approach. Let $VP(P, st)$ denote the weak visibility polygon of P from st, i.e., every point of $VP(P, st)$ is visible from some point of st. Suppose $VP(P, st)$ is known. Is it always possible to locate a set of discrete points $(s = s_o, s_1, ..., s_l = t)$ on st such that $\bigcup_{i=1}^{l} \mathcal{V}(P, s_i) = VP(P, st)$? The boundary of $VP(P, st)$ will always consists of polygonal edges and constructed edges. For example, in Figure 1 the segment ab is a constructed edge of $VP(P, st)$. Observe that the edge ab is visible from only one point, r_4, on st and b, a, c and r_4 are collinear. Hence the point r_4 must be chosen as a viewing point in order to see the entire segment ab. We term such an edge a *degenerate* edge.

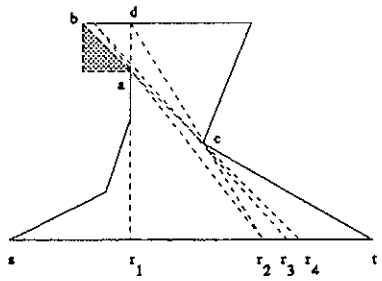


Figure 1: The neighborhood of ab cannot be seen by discrete approximation.

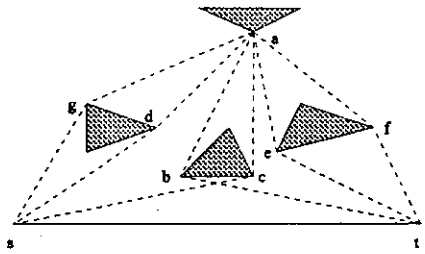


Figure 2: There are two funnels with a as the apex and st as the base.

To understand the effect of these degenerate situations on the discrete visibility approximation, consider a sequence of viewing points that are required to see all points in a neighborhood of a degenerate edge (the edge ab in Figure 1 for example). Suppose r_1 is chosen as the first viewing point. Then the entire region of $VP(P, st)$ except the triangle bad is visible from r_1. In order to see the triangle bad, subsequent viewing points have to be chosen from the interval r_1r_4. Observe that the second viewing point has to be chosen from the interval r_1r_2. Otherwise, the points in the neighborhood of d are not visible from any viewing point on r_2r_4. So, r_2 is chosen as the second viewing point. Analogous argument shows that r_3 has to be chosen as the third viewing point. If the viewing points are chosen in this manner, one needs a sequence of viewing points along r_3r_4 that converges to r_4. Hence, it is not possible to see all points in the neighborhood of ab without the continuous visibility assumption. Thus, it may not be possible to find a discrete approximation to a continuous visibility path in presence of degenerated edges. Therefore, our algorithm stops choosing additional viewing points of a degenerate edge, where the viewing points become very close to each other. However, the presence of such degenerate edges does not necessarily doom the discrete visibility approximation as there is a high probability that the robot may still see the points in the neighborhood of a degenerate edge while moving along some other segment of Q.

3 Funnels

Our analysis and discussion of discrete visibility approximation algorithms are based on the *funnel* concept [2]. The funnel structure inside $VP(P, st)$, with st as the base of all funnels, is defined as follows. For each vertex a, compute all paths inside $VP(P, st)$ from a to both s and t such that every path makes a turn only at a vertex and every path is outward convex (Figure 2). For every path from a to s and t, extend the edge from other end of the edge incident on a till it intersects st . Observe that not all such extensions will intersect st. We are interested only in those paths of a whose extensions have intersected st and such paths of a form the sides of funnels with a as the apex and st as the base. Since every vertex of $VP(P, st)$ is visible from some point on st, two sides of a funnel meet only at the apex. In Figure 2, there are two funnels with a as the apex and st as the base. One funnel is (a, d, s, t, b, a) and the other funnel is (a, c, s, t, e, a). The path (a, f, t) or (a, g, s) does not form a side of a funnel.

Observe that the region enclosed by any funnel does not contain any hole and every vertex of $VP(P, st)$ belong to at least one funnel. For each funnel, the *funnel interval* on st can be located by extending both edges of the funnel incident on the apex to st. Since any point in the interval can

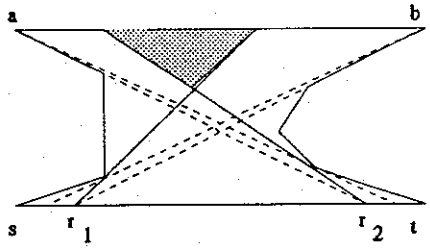


Figure 3: The shaded region is not visible from r_1 and r_2.

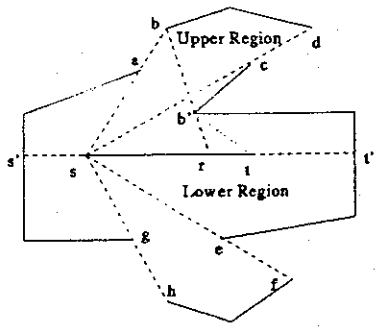


Figure 4: The left and right constructed edges in $\mathcal{V}(P, s)$.

see the entire funnel, the continuous visibility can be replaced by a discrete viewing point located anywhere in the funnel interval. Let a be the apex of a funnel and ab and ad are two edges on the opposite side of the funnel. If a, b and d are collinear, the funnel degenerates and the interval induced by the funnel on st reduces to a point. Therefore, this point must be chosen in order to see the vertex a. Though this point can see the entire region enclosed by the funnel, the point cannot see the neighborhood of the edge ab or ad.

4 Visibility approximation in a known environment

From the description of funnels, it should be clear that it is possible to choose a set of discrete points on st which can at least see all vertices of $VP(P, st)$. This limited form of visibility is achieved by choosing a viewing point on each interval on st induced by all funnels. In practice, this set can be derived by scanning the funnel intervals from s to t by considering only those funnel intervals that do not totally contain any other interval. This approach ensures that the chosen set of discrete points not only can see all vertices of $VP(P, st)$ but also sees the region of $VP(P, st)$ enclosed by all funnels. However, as shown in Figure 3, it is not true that this approach will ensure visibility of all or nearly all of $VP(P, st)$. In this figure, viewing points r_1 and r_2 can see the region enclosed by the funnels with apex a and b but they do not see all points of the polygonal edge ab. It is always be true that points of $VP(P, st)$ lying outside any funnel will form triangles with one edge of the triangle is a part of a polygonal edge. Hence, by choosing one additional point on st for every such triangle, it is possible to see all edges of $VP(P, st)$ which are totally visible from st. We summarize the result in the following theorem.

Theorem 4.1 *Given a segment st inside a known polygon P, the continuous visibility can be approximated by discrete visibility such that (i) the chosen set of viewing points on st sees all vertices and totally visible edges of $VP(P, st)$ and (ii) the size of the set of viewing points is at most twice the number of vertices of $VP(P, st)$.*

5 Visibility approximation in an unknown environment

The above algorithm is based on the assumption that P is known apriori. We now assume that while st is known, the robot has no other prior knowledge of the environment. The goal of the

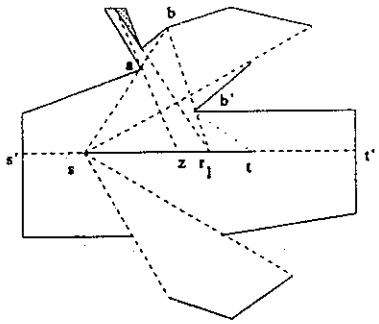


Figure 5: The shaded region is visible from st but not from r_1.

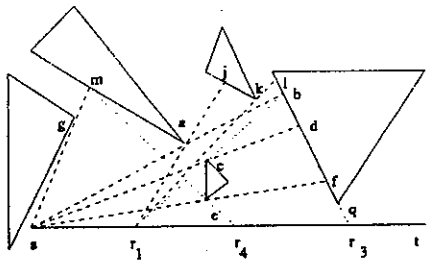


Figure 6: The points r_1, r_3 and r_4 are chosen for exploring funnels F_1, F_3 and F_4.

discrete approximation algorithm in this case is to find a set of discrete viewing points on st that gives a good approximation of $VP(P, st)$. Assume that $\mathcal{V}(P, s)$ has been computed. Note that $\mathcal{V}(P, s)$ does not contain any hole. Let s' and t' be the points on the boundary of $\mathcal{V}(P, s)$ such that s and t lie on the segment $s't'$ (Figure 4). We call the region bounded by the clockwise boundary of $\mathcal{V}(P, s)$ from s' to t' and the segment $s't'$ as the *upper region* of st. Similarly, we call the region bounded by the clockwise boundary of $\mathcal{V}(P, s)$ from t' to s' and the segment $s't'$ as the *lower region* of st. As mentioned earlier, the boundary of $\mathcal{V}(P, s)$ consists of polygonal edges and constructed edges. One endpoint of a constructed edge is always a vertex whereas other endpoint is not a vertex. If the vertex of a constructed edge in encountered as the first point (respectively, the last point) of the constructed edge while traversing the boundary of $\mathcal{V}(P, s)$ in clockwise order, then we call the constructed edge as a *left* (respectively, *right*) constructed edge. In Figure 4, ab and ef are left constructed edges in $\mathcal{V}(P, s)$ and, cd and gh are right constructed edges in $\mathcal{V}(P, s)$. Note that a, c, e and g are vertices of P. Let s be the first point w_1 of W. We now wish to choose the next point w_2 of W on st. We need the following lemma.

Lemma 5.1 *For any point $z \in VP(P, st) - \mathcal{V}(P, s)$, the line segment zr inside $VP(P, st)$, where $r \in st$, must intersect either a left constructed edge of the upper region of st in $\mathcal{V}(P, s)$ or a right constructed edge of the lower region of st in $\mathcal{V}(P, s)$.*

The above lemma suggest that the next point w_2 on st must be chosen to view those particular types of constructed edges of $\mathcal{V}(P, s)$ in $VP(P, st)$. Let ab be a left constructed edge in the upper region of st in $\mathcal{V}(P, s)$, where a is a vertex of P. Let b' be the next vertex of b in the Euclidean shortest path from b to t inside $\mathcal{V}(P, s)$ (Figure 4). Extend bb' from b' to meet st at a point r. Note that r will be the point t if the segment bt is the Euclidean shortest path from b to t. The entire constructed edge ab can be seen from any point on the interval sr. The algorithm computes such a point r for every left constructed edge in the upper region and for every right constructed edge of the lower region. Let r_1 be the closest point to s among all these points. The interval sr_1 is called the *smallest interval* of s. So the robot moves from s to r_1 and computes $\mathcal{V}(P, r_1)$. $\mathcal{V}(P, r_1)$ now has a new set of constructed edges. Then, for every left constructed edge in the upper region and for every right constructed edge in the lower region, the corresponding interval on r_1t is located. Let r_1r_2 be the smallest interval among all these intervals. So the robot moves from r_1 to r_2 and computes $\mathcal{V}(P, r_2)$. This process is repeated till t is reached.

We now discuss the issue whether or not the chosen set of discrete points can see all vertices and totally visible polygonal edges of $VP(P, st)$. Consider Figure 5. Let sr_1 be the smallest interval on st induced by the left pocket ab of the upper region in $\mathcal{V}(P, s)$. So the robot has moved from s to r_1 and then it has computed $\mathcal{V}(P, r_1)$. Observe that any point in the shaded region is not visible from r_1 but all points in the shaded region are visible from some point z on sr_1. As robot will move towards t from r_1, this shaded region cannot be seen from the subsequent points chosen from r_1t. Hence, this example shows that our algorithm does not guarantee that all vertices and totally visible polygonal edges in $VP(P, st)$ can be seen from the chosen set of discrete points. It seems that for any choice of discrete points on st, we can always construct such an example to show that all vertices and totally visible edges in $VP(P, st)$ are not visible from the chosen points.

The above approximate algorithm is based on the assumption that the robot is expected to move in one direction on st like a human watchman. If the robot is allowed to retrace the path and is also allowed to choose some more points, then points like z in Figure 5 can be chosen to explore the hidden region of $VP(P, st)$. We now present an approximation algorithm that allows backtracking. Assume that $\mathcal{V}(P, s)$ has been computed. Let ab be a constructed edge where a is a vertex. Construct the funnel with b as the apex and st as the base. So one side of the funnel consists of segments sa and ab and the other side is the Euclidean shortest path from b to t inside $\mathcal{V}(P, s)$. If it is not a degenerated funnel, then the algorithm locates the point r on st by extending bb' from b' to st as shown in Figure 4, where bb' is the edge incident on b' in the Euclidean shortest path from b to t inside $\mathcal{V}(P, s)$. Then the algorithm computes $\mathcal{V}(P, r)$. Let cd be a constructed edge in $\mathcal{V}(P, r)$, where c is a vertex, such that the segment dr intersects ab. We call cd as a constructed edge created in the hidden region of ab. For all those constructed edges cd of $\mathcal{V}(P, r)$ which are created in the hidden region of ab, construct the funnel with apex as d and st as the base. Note that r can see both sides of these funnels and therefore all these funnels lie inside $\mathcal{V}(P, r)$. For each non-degenerated funnel, the algorithm locates the interval on st induced by the funnel and computes the visibility polygon from that endpoint of the interval which is not r. Note that one endpoint of the intervals induced by these funnels is r. Process is repeated till all are degenerated funnels or the intervals induced by funnels on st are very small.

To summarize the general procedure, the algorithm first constructs funnels F_1, F_2,... in $\mathcal{V}(P, s)$ where one edge incident on the apex is a constructed edge of $\mathcal{V}(P, s)$. Then the algorithm removes the degenerated and potential degenerated funnels from the list of funnels for further exploration. From each remaining funnel F_i in the list, the algorithm locates the interval induced by the funnel on st and chooses the appropriate endpoint of the interval as the next viewing point to view the hidden region of the constructed edge associated with F_i. Then it generates funnels $F_{i,1}$, $F_{i,2}$,... with respect to each constructed edge of the visibility polygon computed from the current viewing point and prunes the list by removing all degenerated and potential degenerated funnels from the list, and then recursively explore each of the remaining funnels in the list. The algorithm terminates when the list of funnels for further exploration in each recursive call becomes empty. In Figure 6, there are four constructed edges ab, cd, ef and gm in $\mathcal{V}(P, s)$. So there are four funnels with st as the base, where b, d, f and m are the apexes of the funnels $F_1 = (b, a, s, t, e, c, b)$, $F_2 = (d, c, s, t, e, c, d)$, $F_3 = (f, e, s, t, q, f)$ and $F_4 = (m, g, s, t, e, m)$ respectively. Since F_2 is a degenerated funnel, it is removed from the list. The points r_1, r_3 and r_4 are chosen as the next viewing points to view the hidden region associated with funnels F_1, F_3 and F_4 respectively.

Let us discuss how the algorithm decides to declare a funnel as a potential degenerated funnel. We assume that there is a lower bound, δ, on the distance between successive viewing points. If the interval induced by a funnel becomes very small (i.e. less that δ), it is declared as a potential

degenerated funnel. Suppose the interval induced by a funnel is very small but the last few views could not see a new vertex but only the apex point of the funnel has been shifted as in Figure 1, the funnel is declared as a potential degenerated funnel. This conditions for potential degenerated funnels ensure that total number of views computed by the algorithm is proportional to the sum of the sides of all funnels which can be at most the square of the number of vertices in $VP(P, st)$. Once all funnels are constructed, one additional viewing points may be required to see each polygonal edge which is totally visible from st as discussed earlier. Since there could be some vertices hidden in the region associated with potential degenerated funnels, there is no guarantee that all vertices and totally visible polygonal edges of $VP(P, st)$ can be seen from the chosen set of points as in the case of our 'off-line' algorithm. However, it appears that in most situation, our 'on-line' backtracking algorithm will succeed to see all vertices and totally visible polygonal edges of $VP(P, st)$. We summarize the result in the following theorem.

Theorem 5.1 *Given a segment st inside an unknown polygon P, the continuous visibility can be approximated by discrete visibility such that (i) the chosen set of viewing points on st can see a large portion of $VP(P, st)$ and (ii) the size of the set of viewing points is at most the square of the number of vertices of $VP(P, st)$.*

6 Concluding remarks

Above approximation algorithms are useful in designing exploration algorithms as follows. Suppose a convex robot C wants to explore the free-space of a polygon P under the restriction that the robot computes visibility polygons only at a discrete number of points on its path. If C knows P apriori, then the configuration space of P can be computed using Minkowski sum of C and P. Note that the configuration space may not cover the entire P. Then by treating each non-polygonal edge of the configuration space as a given line segment st, a set of viewing points on st can be chosen using the algorithm in Section 4 so that the robot can take views from these points in order to see the free-space of P lying outside the configuration space. So, the exploration problem for a convex robot reduces to the problem of computing visibility polygons from the chosen set of viewing points. If C does not know P apriori, the configuration space of P cannot be computed using Minkowski sum. Ghosh and Burdick [1] have designed an algorithm for this situation which computes the configuration space incrementally. Then, each non-polygonal edge of the configuration space can be discretized by the approximation algorithm in Section 5 so that the robot can take views from the chosen set of viewing points.

7 References

1. S. K. Ghosh and J. W. Burdick, *Exploring an unknown polygonal environment with a sensor based strategy*, manuscript, 1997.

2. S. K. Ghosh and D. Mount, *An output sensitive algorithm for computing visibility graphs*, SIAM Journal on Computing, vol. 20 (1991), 888-910.

3. S. K. Ghosh and S. Saluja, *Optimal on-line algorithms for walking with minimum of turns in unknown streets*, Computational Geometry: Theory and Applications, vol. 6 (1997), 1-27.

Why Taro Can Do Geometry

Jin Akiyama
Math. Inst., Tokai University

Abstract

Television is a powerful educational medium. It allows one to reach a huge number of students (5 million in Japan) in many different locations simultaneously. In the last seven years, I have had the opportunity of using this medium to teach mathematics and to foster an appreciation of mathematics.

In this talk, I will be presenting film clips from various geometry lessons aimed at either elementary school or junior high school students, while commenting on the contents and objectives of the lessons and the impact the lessons have made on the Japanese audience, judging from the responses received from parents and students around the country.

A quantum-searching application note

(Preliminary report)

Ngọc-Minh Lê *

Abstract

We observe that the quantum search algorithm of Grover [4], its improvement by Boyer et al [1], and the quantum algorithm for finding the minimum of Dürr and Høyer [3], when applied to some typical problems in computational geometry, curiously outperform the known efficient classical algorithms.

To perform computations, quantum computing exploits properties of microscopic systems whose behaviour obeys quantum mechanics. The basic principles of quantum mechanics are summarized below.

1 Quantum computing

A unitary vector space U is a complex vector space endowed with a complex scalar product. In quantum mechanics, the elements of U are denoted $|u\rangle, |\varphi\rangle, \ldots$ in place of the usual notations $\vec{u}, \vec{\varphi}, \ldots$. The scalar product of any two vectors $|u\rangle$ and $|v\rangle$ is denoted by $\langle u|v\rangle$. A linear operator $\mathcal{L}$ on U is a linear mapping from U to itself. The adjoint operator of $\mathcal{L}$ is an operator on U, denoted $\mathcal{L}^*$, so that $\langle u|\mathcal{L}v\rangle = \langle \mathcal{L}^*u|v\rangle$ holds for every $|u\rangle, |v\rangle \in U$.

A complex number Λ is an eigenvalue of $\mathcal{L}$ iff $\Lambda|u_\Lambda\rangle = \mathcal{L}|u_\Lambda\rangle$ for some $|u_\Lambda\rangle \in U$. In general, eigenvalues need not be real. However, if $\mathcal{L} = \mathcal{L}^*$, then the eigenvalues of $\mathcal{L}$ are real. An operator $\mathcal{L}$ on U is a unitary transformation iff its inverse equals its adjoint operator: $\mathcal{L}^{-1} = \mathcal{L}^*$.

In microscopic systems, the measuring process disturbs the system state in a non-negligible way. Quantum mechanics models the measurable properties of a system by self-adjoint operators on a unitary space—the *observables* of the system. The state of the system is fully characterized by a unit-length vector $|\Phi\rangle$ of the unitary space, which is required to be spanned by the normalised eigenvectors of the observable, i.e., $|\Phi\rangle = \sum_\Lambda \Phi(\Lambda)|u_\Lambda\rangle$, where $\Phi(\Lambda) = \langle u_\Lambda|\Phi\rangle$. The possible results of any measurement of an observable $\mathcal{L}$ are exactly its eigenvalues. One fundamental statement of quantum mechanics is that the probability for the occurrence of the eigenvalue Λ in a measurement of $\mathcal{L}$ is given by $w_\Lambda = |\langle u_\Lambda|\Phi\rangle|^2$, and that, after the measurement, the system will be in the state $|u_\Lambda\rangle$. The complex number $\langle u_\Lambda|\Phi\rangle$ is called the *probability amplitude* for Λ.

To describe a quantum system that is composed of several independent subsystems, for simplicity, U^1 and U^2, we use the product space $U = U^1 \times U^2$, which is spanned by the direct products of the basis vectors of U^1 and U^2: $|u^1_\Lambda u^2_M\rangle = |u^1_\Lambda\rangle|u^2_M\rangle$, with the scalar product $\langle u^1u^2|v^1v^2\rangle = \langle u^1|v^1\rangle\langle u^2|v^2\rangle$, for any $u^i, v^i \in U^i$.

A *quantum bit* or, for short, *qubit* is a state vector of a quantum system whose state space is 2-dimensional. An example of a qubit is the spin state of an electron, which is given by $|\Phi\rangle = \alpha|\frac{1}{2}\rangle + \beta|-\frac{1}{2}\rangle$ (with $|\alpha|^2 + |\beta|^2 = 1$).

*Author's address: EsHag Research, Rönselstraße 17, D-58135 Hagen, Germany.

A *quantum register* of length L is a quantum system composed of L qubits. If we use $|0\rangle$ and $|1\rangle$ to denote the basis vectors of the state space for a qubit, then the basis vectors of the state space for the quantum register is given by $|b_1 \cdots b_L\rangle = |b_1\rangle \cdots |b_L\rangle$, where $b \in \{0,1\}$. Clearly the general state vector of a quantum register of length L is $|\Phi\rangle = \sum_{i=0}^{N-1} \alpha_i |i\rangle$, where $|i\rangle$ corresponds to the binary representation of the number i, and $N = 2^L$.

Computing on a quantum computer consists in initializing the register with some input, sequentially applying appropriate unitary tranformations to the state space of the quantum register, and finally obtaining the output by observing the register.

The most important quantum algorithm is given in [8]. In this report we deal with quantum algorithms that search among a set of given items to find one that satisfies a particular property.

2 Quantum searching

The search problem is as follows. Let $X_N = \{0, 1, \ldots, N-1\}$ for some integer N. Given any function $F : X_N \to \{0,1\}$, find $i \in X_N$ so that $F(i) = 1$. Algorithms on a classical computer require $O(N)$ time to find such an i (if it exists). Recently, Grover has discovered an algorithm for the quantum computer to solve the search problem in only $O(\sqrt{N})$ expected time [4]. The Grover's algorithm is described below (we follow [1].)

Assume that $N = 2^L$; see [1] for the general case. The conditional phase shift transform corresponding to F is defined by

$$S_F|i\rangle = \begin{cases} -|i\rangle & \text{if } F(i) = 1 \\ |i\rangle & \text{otherwise.} \end{cases}$$

In particular, denote S_{F_0} by S_0, where $F_0(i) = 1$ iff $i = 0$. Next, we need the Walsh-Hadamard transform, which is the unitary transform

$$W|j\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} (-1)^{i \cdot j} |i\rangle,$$

where $i \cdot j$ is the number of 1 in the bitwise AND of i and j. The basic ingredient of the Grover's algorithm is the unitary transform obtained by composing the transforms above:

$$G_F = -WS_0WS_F.$$

QUANTUM SEARCHING ALGORITHM

1. Initialize the register to the state

$$|\Phi\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle.$$

2. Apply $O(\sqrt{N})$ times the unitary transform G_F to the register.

3. Observe the register. If there is a unique i so that $F(i) = 1$, then the measurement yields the state $|i\rangle$ with a probability $\geq \frac{1}{2}$.

The algorithm above has to assume that there is exactly one i so that $F(i) = 1$. It has been improved by Boyer et al to allow the general case [1]. We next consider an application of this algorithm.

Detecting intersection among spheres. Given a set of n spheres in 3-space, determine if any two of these spheres intersect each other. The algorithm in [5] solves this problem in $O(n \log^2 n)$ time.

To apply the quantum search algorithm to this problem, consider the function $F : X_n \times X_n \to \{0, 1\}$ defined by setting $F(i, j) = 1$ if and only if the spheres i and j intersect, with $i \neq j$. Note that the domain of F consists of pairs (i, j). In addition, the register needed in the algorithm will be composed of two registers, each of length $\log n$. (Assume for simplicity that $n = 2^l$ for some integer l.) The algorithm will start from the state

$$|\Phi\rangle = \frac{1}{n} \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} |i\rangle |j\rangle.$$

Thus, the algorithm needs $O(n)$ expected time to find a pair of intersecting spheres, with high probability. (We assume that $F(i, j)$ can be evaluated in constant time, for all i, j.)

3 Quantum search of the minimum

Given a table of N items, each of which holds a value from an ordered set, the minimum searching problem is to find an item holding the smallest value. Thus, to each $i = 0, \ldots, N-1$ a value $T(i)$ is associated. We want to find an i so that $T(i)$ is minimal.

Dürr and Høyer give a quantum minimum searching algorithm that runs in $O(\sqrt{N})$ expected time to find the minimum with high probability [3]. Their algorithm is based on the algorithm of Grover [4] and Boyer et al [1], refer to the previous section for a short description. To find the minimum, the algorithm calls the quantum searching algorithm a number of times, each time with an appropriate F that depends on an index in the table T. This function is defined as follows. For every $y = 0, \ldots, N-1$, define a function F_y by

$$\forall i : 0 \leq i \leq N-1, F_y(i) = 1 \text{ if and only if } T(i) < T(y).$$

Call y a *threshold index.*

QUANTUM MINIMUM SEARCHING ALGORITHM

1. Choose a threshold index y uniformly at random from $\{0, \ldots, N-1\}$.
2. Repeat the following until the total number of Grover iterations that are applied is more than $30\sqrt{N}$.
   (a) Call the quantum searching algorithm to find an index y' with $F_y(y') = 1$.
   (b) If such an index is found, then set the current threshold index to that index.
3. Return the threshold index, y. The minimum is $T(y)$ with a probability of at least $\frac{1}{2}$.

We now apply the algorithm to a problem in computational geometry.

The closest pair of lines in 3-space. Given a set of n lines in 3-space, we want to find the shortest segment connecting two lines. The algorithm in [6] solves this problem in $O(n^{5/3+\epsilon})$ randomized expected time, for any $\epsilon > 0$.

To solve the problem using the quantum algorithm for finding the minimum, we define $T(i,j)$ to be the distance between the lines i and j, for $0 \leq i,j \leq n-1$. The index into the table T is thus a pair of indices. For simplicity assume that $n = 2^l$, for some integer l. We let the quantum algorithm operate on a pair of registers, each of length $\log n$, so that it will start from the state

$$|\Phi\rangle = \frac{1}{n} \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} |i\rangle|j\rangle.$$

Thus, the algorithm needs $O(n)$ expected time to find the closest pair of lines, with high probability. (We assume that $T(i,j)$ can be evaluated in constant time, for all i,j.)

4 Discussions

Counting particular objects among a set of objects is also a common problem in computational geometry [7]. There is a quantum counting algorithm described in [1] that can be applied to that kind of problem.

We have applied the quantum search algorithm to some typical problems in computational geometry in a very simple way, without making much effort to exploit the structure of the problem. Nevertheless, the power of quantum searching suffices to outperform the known efficient algorithms for classical computers.

Acknowledgements I wish to thank C. Dürr for clarifying some points of the paper [3].

References

[1] M. Boyer, G. Brassard, P. Høyer, and A. Tapp: *Tight bounds on quantum searching.* Proc. 4th Workshop Physics and Computation, pp. 36–43, 1996. Also available from the Los Alamos archive: `http://xxx.lanl.gov/archive/quant-ph/9605034`, 1996.

[2] G. Brassard: *Searching a quantum phone book.* Science, Vol. 275, 31 Jan. 1997, 627–628.

[3] C. Dürr and P. Høyer: *A quantum algorithm for finding the minimum.* Preprint, Los Alamos archive: `http://xxx.lanl.gov/archive/quant-ph/9607014`, 1996.

[4] L. K. Grover: *A fast quantum mechanical algorithm for database search.* Proc. 28th Ann. ACM Symp. Theory of Computing, 212–219, 1996.

[5] J. E. Hopcroft, J. T. Schwartz, and M. Sharir: *Efficient detection of intersections among spheres.* Int. J. Robotics Research, Vol. 2, No. 4, 77–80, 1983.

[6] M. Pellegrini: *On collision-free placements of simplices and the closest pair of lines in 3-space.* SIAM J. Comput., Vol. 23, No. 1, pp. 133–153, February 1994.

[7] M. Pellegrini: *On counting pairs of intersecting segments and off-line triangle range searching.* Algorithmica(1997) 17: 380–398.

[8] P. W. Shor: *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer.* SIAM J. Computing. To appear.

Some Methods to Determine the Sign of a Long Integer from its Remainders

Toshiyuki Imai
Department of Mathematical Engineering and Information Physics
University of Tokyo

1 Introduction

In geometric algorithms, we can obtain the combinatorial structures of geometric objects by various sign-tests of the results of arithmetics such as addition, subtraction and multiplication. These arithmetics are done exactly, because numerical errors in arithmetics may cause failure in executing programs. Moreover, exact arithmetics are required in techniques to treat non-degenerate inputs such as symbolic perturbation [2, 3, 4].

In general, the result of arithmetics is quite longer even if input data have bounded length (e.g. 32 bit integer etc.) It increases the execution time of the implemented program.

If the execution time for the unit length is O(1) in addition, subtraction and multiplication, that for the m-ple length may be $O(m)$ in addition and subtraction and $O(m^2)$ in multiplication.

By the Fast Multiplication method[1, 5], we can reduce the time complexity of multiplication to $O(m \log m \log\log m)$, but it is practically not fast for the geometric purpose. It is said the naive method is faster if m is less than several hundred.

Modular arithmetic is one alternative, which does addition, subtraction or multiplication in $O(m)$ time. It is also good for parallel computing. One difficulty to use modular arithmetic is to reconstruct the integer from its remainders. It takes $O(m(\log m)^2 \log\log m)$ and the algorithm contains the Fast Multiplication of long integers[1].

To obtain combinatorial structures, only the signs of integers are required and the values are not. In this paper, we present 4 methods to determine the sign of a long integer from its remainders by using modular arithmetics of the unit length. They take $O(m^2)$ time but they are relatively simple and practically fast.

2 Notations, Assumptions, Properties and the Problem

Assume n be the unit length of integer. It means the unsigned integer of the unit length is between 0 and $2^n - 1$. We assume the computer can show us the following information:

- whether the result of addition is in the unit length or not,
- the lower n bits of the result of multiplication and the other upper bits.

In this sense, the methods in this paper depend on the hardware but this assumption holds in almost all computers.

To describe modular arithmetics, we introduce 2 notations: for $A \in \mathbf{Z}$, $B \in \mathbf{N}$, let $[A/B]$ be the integral quotient and let $A\%B$ be the remainder of A divided by B. Then $0 \leq A\%B < B$ and the following basic formula holds:

$$A = [A/B]B + A\%B. \tag{1}$$

Let $p_0, \cdots, p_m$ be relatively prime positive integers which satisfies the following properties:

$$p_i = 2^n - \alpha_i,\ \alpha_i \geq 0,\ p_i > {\alpha_i}^2. \tag{2}$$

Let $p = p_1 \cdots p_m$. Obviously, $m \leq 2^{n/2}$. This means we have little meaning to discuss the theoretical performance of the algorithm as $m \to \infty$. In practical sense, however, m can be large enough.

We can do modular arithmetics as follows:

Assume Z is the result of an integral arithmetic, and let Z_L be the lower n bits and Z_U be the other upper bits (i.e. $Z = 2^n Z_U + Z_L$). Since $2^n \equiv \alpha_i \bmod p_i$, $Z \equiv Z_L + \alpha_i Z_U \bmod p_i$. Iterate the following procedure until $Z_U = 0$:

$$Z_L \leftarrow \text{lower } n\text{bit of } Z;\ Z_U \leftarrow \text{upper bits of } Z;\ Z \leftarrow Z_L + \alpha_i Z_U. \tag{3}$$

It is known that the iteration terminates in 3 times if $p_i > \alpha_i^2$. After the iteration, the result of modular arithmetic $Z\%p_i$ is Z if $Z < p_i$, or $Z - p_i$ otherwise.

We formulate the problem as follows:

Problem:
For several given integers X_i ($0 \leq X_i < p_i$), determine the sign of the integer X' ($-p/2 \leq X' < p/2$) s.t. $X' \equiv X_i \bmod p_i$ only by arithmetics for integers of the unit length.

Let $X = X'\%p$ and $X_i = X\%p_i$ holds. The following properties hold:

$$X' \geq 0 \quad \Leftrightarrow \quad X < p/2, \tag{4}$$
$$X' < 0 \quad \Leftrightarrow \quad X \geq p/2. \tag{5}$$

Then we can obtain the sign of X' by comparing X and $p/2$.

3 Method A

Let $p_m = 2^n$. We use $p_1, \cdots, p_m$ as moduli. $p = p_1 \cdots p_m$ is almost 2^{mn}.
The following properties hold

$$\begin{aligned} X' \geq 0 \quad &\Leftrightarrow \quad X < p/2 = p_1 \cdots p_m/2 && (6)\\ &\Leftrightarrow \quad [X/p_1] < p_2 \cdots p_m/2 && (7)\\ &\Leftrightarrow \quad [[X/p_1]/p_2] < p_3 \cdots p_m/2 && (8)\\ &\cdots \\ &\Leftrightarrow \quad [\cdots [[X/p_1]/p_2] \cdots /p_{m-1}] < p_m/2. && (9) \end{aligned}$$

The last inequality is the only one we can test in the unit length. This means that if we can compute the value of $[\cdots [[X/p_1]/p_2] \cdots /p_{m-1}]$, we obtain the sign of X'.

For each k, let $Q_k = [\cdots [[X/p_1]/p_2] \cdots /p_{k-1}]$.

$$Q_{k+1} = [Q_k/p_k],\ Q_1 = X \tag{10}$$

hold and $Q_m = [\cdots [[X/p_1]/p_2] \cdots /p_{m-1}]$. So we obtain the sign of X' if we can get the value of Q_m.

By the definition,

$$\begin{aligned} Q_k &= [Q_k/p_k]p_k + Q_k\%p_k && (11)\\ &= Q_{k+1}p_k + Q_k\%p_k && (12) \end{aligned}$$

holds. By describing this equation modulo p_j ($j = k+1, \cdots, m$), we have

$$Q_k\%p_j \equiv (Q_{k+1}\%p_j)p_k + Q_k\%p_k \bmod p_j. \tag{13}$$

Then, we have

$$Q_{k+1}\%p_j \equiv q_{kj}(Q_k\%p_j - Q_k\%p_k) \bmod p_j, \tag{14}$$

where each q_{kj} is an integer satisfying $p_k q_{kj} \equiv 1 \bmod p_j$, which can be computed in the preprocess. We transform this congruence into recursive formula:

$$Q_{k+1}\%p_j = (q_{kj}(Q_k\%p_j - Q_k\%p_k))\%p_j \ (j = k+1, \cdots, m). \tag{15}$$

From this formula, we can compute Q_m ($= Q_m\%p_m$) from X_i ($= Q_1\%p_i$) ($i = 1, \cdots, m$) recursively. and finally we obtain the sign of X'.

To illustrate the procedure, we identify $Q_k\%p_j$ with (k, j). This formula shows we can obtain $(k+1, j)$ from both (k, j) and (k, k). We illustrate this formula as

$$\begin{matrix} (k+1, j) & \leftarrow (k, j) \\ & \nwarrow (k, k). \end{matrix} \tag{16}$$

We can obtain (m, m) from $(1,1), (1,2), \cdots, (1, m)$ as Figure 1.

Time and Space complexities are both $O(m^2)$ in the worst case. We can terminate the procedures in method A when $Q_k\%p_k = Q_k\%p_j$ ($j = k+1, \cdots, m$) holds (i.e. $Q_m = Q_k\%p_k$ holds).

While the method A computes integral quotients Q_k ($= [X/p_1 \cdots p_{k-1}]$) one by one, the next method computes reminders $X\%(p_1 \cdots p_k)$ successively.

$$\begin{array}{llllll}
(m,m) & \leftarrow (m-1,m) & \leftarrow (m-2,m) & \cdots & \leftarrow (1,m) \\
 & \nwarrow (m-1,m-1) & \leftarrow (m-2,m-1) & \cdots & \leftarrow (1,m-1) \\
 & & \nwarrow (m-2,m-2) & \cdots & \leftarrow (1,m-2) \\
 & & & \ddots & \vdots \\
 & & & & \nwarrow (1,1)
\end{array}$$

Figure 1: Procedures in Method A and B

4 Method B

Let $p_m = 2^n$ and we also use $p_1, \cdots, p_m$ as moduli. $p = p_1 \cdots p_m$ is almost 2^{mn}. Let $p_{1j} = p_1 \cdots p_{j-1}$ for $j \geq 2$, and $p_{11} = 1$.

We define $Y_1, \cdots, Y_{m+1}$ by $Y_j = X\%p_{1j}$ for $j \geq 2$ and $Y_1 = 0$. The following properties hold:

$$Y_j \equiv X_i \bmod p_i \ (i = 1, \cdots, j-1),\ 0 \leq Y_j < p_{1j}. \tag{17}$$

If we can compute the value of $[X/p_{1m}]$, we obtain the sign of X' because

$$\begin{aligned} X' \geq 0 &\Leftrightarrow X < p/2 = p_{1m}p_m/2 &(18)\\ &\Leftrightarrow [X/p_{1m}] < p_m/2. &(19)\end{aligned}$$

holds. From the basic formula $X = [X/p_{1m}]p_{1m} + Y_m$, we have a congruence:

$$X \equiv [X/p_{1m}]p_{1m} + Y_m \bmod p_m \tag{20}$$

and it can be transformed to

$$[X/p_{1m}] \equiv c_m(X_m - Y_m) \bmod p_m \tag{21}$$

where c_j is an integer satisfying $c_j p_{1j} \equiv 1 \bmod p_j$, which can be computed in the preprocess.

Because $0 \leq [X/p_{1m}] < p_0$,

$$[X/p_{1m}] = (c_m(X_m - Y_m))\%p_m \tag{22}$$

holds. This means if we can compute the value of $Y_m\%p_m$, we obtain the sign of X'.

Since $Y_j \equiv X_i \bmod p_i \ (i = 1, \cdots, j-1)$ for $j \geq 2$, $X - Y_j \equiv 0 \bmod p_i \ (i = 1, \cdots, j-1)$ holds. This means $X - Y_j$ is a multiple of p_{1j} because each p_i is relatively prime. So we have

$$X - Y_j = ap_{1j}. \tag{23}$$

Making this equation the congruence modulo p_j and transforming it, we have

$$\begin{aligned} a &\equiv c_j(X_j - Y_j) &(24)\\ &\equiv c_j(X_j - Y_j\%p_j) \bmod p_j. &(25)\end{aligned}$$

Let $A = (c_j(X_j - Y_j\%p_j))\%p_j$, and

$$Y_{j+1} = Y_j + Ap_{1j} \tag{26}$$

(Note that $0 \leq A < p_j$). This formula holds for not only $j \geq 2$ but $j = 1$.

By taking remainder of each side modulo p_k $(k = j+1, \cdots, m)$, we have

$$Y_{j+1}\%p_k \equiv Y_j\%p_k + ((c_j(X_j - Y_j\%p_j))\%p_j)p_{1j} \bmod p_k \ (k = j+1, \cdots, m). \tag{27}$$

In the preprocess, we compute $p_{1j}\%p_k$ and c_j.

We can consider this congruence as a recursive equation to get $Y_{j+1}\%p_k$ from $Y_j\%p_k$ and $Y_j\%p_j$ The initial value $Y_1\%p_k = 0$ $(k = 1, \cdots, m)$ since $Y_1 = 0$. Now we can compute $Y_m\%p_m$ by this recursive equation and we obtain the sign of X'.

To illustrate this procedure, we identify $Y_j\%p_k$ with (j,k). The recursive equation shows that we can get $(j+1,k)$ from (j,j) and (j,k). We illustrate this as follows:

$$\begin{array}{ll} (j+1,k) & \leftarrow (j,k) \\ & \nwarrow (j,j). \end{array}$$

We can compute (m, m) from $(1,1), \cdots, (1, m)$ as the method A and we obtain the sign of X'. We can see the procedures in Figure 1 at the previous section.

Time complexity is $O(m^2)$ and space one is $O(m)$ in worst case. We can terminate the procedure if $Y_j \equiv X_k \bmod p_k$ $(k = j, \cdots, m)$ since we find $Y_m = Y_j$, which means Y_m is small $(Y_m < p_{1j})$.

The both methods A and B terminate early when X is small. To apply these method to $-X'$, they terminate early when $p - X$ is small. So we can say the method A and B terminate early when the absolute value $|X'|$ is is small. The next 2 methods terminates when X' is large.

5 Method C

Let $p_0 = 2^n$ and we use $p_0, \cdots, p_m$ as moduli. Let $P = p_0 \cdots p_m$ and $p = p_1 \cdots p_m (= P/p_0)$. P is almost $2^{(m+1)n}$. In this section we assume X' is in the range of $-P/2 \leq X' < P/2 - mp$. We can easily see X' is in this range if $-p/2 \leq X' < p/2$ (note that $m \leq 2^{n/2}$).

$$X' \geq 0 \quad \Leftrightarrow \quad X < p_0 p/2 \tag{28}$$

$$\Leftrightarrow \quad [X/p] < p_0/2 \tag{29}$$

holds. From the basic formula $X = [X/p]p + X\%p$, we have

$$X_0 \equiv [X/p]p\%p_0 + (X\%p)\%p_0 \bmod p_0. \tag{30}$$

This means we get the value $[X/p]$ if we can compute $(X\%p)\%p_0$.

From the Chinese Reminder Theorem [1, 5], we have

$$X\%p \equiv ((X_1 u_1)\%p_1)q_1 + ((X_2 u_2)\%p_2)q_2 + \cdots + ((X_m u_m)\%p_m)q_m \bmod p. \tag{31}$$

where $q_i = p/p_i$ and $u_i q_i \equiv 1 \bmod p_i$. Let $Y_i = (X_i u_i)\%p_i$, and we have

$$X\%p \equiv Y_1 q_1 + Y_2 q_2 + \cdots + Y_m q_m \bmod p. \tag{32}$$

Let Y be the right side of the congruence: $Y = Y_1 q_1 + Y_2 q_2 + \cdots + Y_m q_m$.

We evaluate how large Y is. Since $0 \leq Y_i < p_i$, $p_i q_i = p$, we have

$$Y = Y_1 q_1 + Y_2 q_2 + \cdots + Y_m q_m < p + p + \cdots + p = mp. \tag{33}$$

This means that $X\%p$ and Y has the same remainder divided by p and that their difference is less than mp. We take Y as an approximation of $X\%p$. Since $X\%p = Y\%p$, from the difference of the each side of the basic formule $X = [X/p]p + X\%p$ and $Y = [Y/p]p + Y\%p$, we have

$$X - Y = ([X/p] - [Y/p])p. \tag{34}$$

Making this equation into the congruence modulo p_0 and transforming it, we have

$$[X/p] \equiv c(X_0 - Y) + [Y/p] \bmod p_0. \tag{35}$$

where c is an integer satisfying $cp \equiv 1 \bmod p_0$, which can be computed in the preprocess. From the formula 33, we can see $0 \leq [Y/p] \leq m - 1$. Then we have the following properties:

$$(c(X_0 - Y))\%p_0 < p_0/2 \quad \Rightarrow \quad [X/p] < p_0/2, \tag{36}$$

$$(c(X_0 - Y) + m - 1)\%p_0 \geq p_0/2 \quad \Rightarrow \quad [X/p] \geq p_0/2. \tag{37}$$

Now we can see that we obtain the sign of X' in these 2 cases.

In the other case: $(c(X_0 - Y))\%p_0 \geq p_0/2$ and $(c(X_0 - Y) + m - 1)\%p_0 < p_0/2$, we have

$$-mp < X' < (m-1)p. \tag{38}$$

Let m' be the greatest integer which satisfies $m' < 2^{(n-1)/2}$. From the definitions of m and m', $m'm < 2^{(n-1)}$ holds. So we have $-p_0 p/2 = -2^{n-1}p \leq -m'mp < m'X' < m'(m-1)p \leq 2^{n-1}p - mp = p_0 p/2 - mp$. This means we can apply this method to $m'X'$ instead of X'. Since $m'X' \equiv m'X_i \bmod p_i$, To apply this method to $m'X'$, we have only to do $X_i \leftarrow (m'X_i)\%p_i$. If we do not determine the sign again, we iterate this until we obtain the sign of X'. The iteration is terminated in $3m + 1$ times because we can easily see $m'^{3m+1} > mp$.

Time complexity is $O(m^2)$ and space one is $O(m)$ in worst case. The iteration will terminate early when $|X'|$ is large.

6 Method D

Let $p_0 = 2^n$ and we use $p_0, \cdots, p_m$. Let $P = p_0 \cdots p_m$ and $p = p_1 \cdots p_m (= P/p_0)$. P is almost $2^{(m+1)n}$. In this section we assume X' is in the range of $-P/2 \leq X' < P/2 - mp$. We can easily see X' is in this range if $-p/2 \leq X' < p/2$. As we see in the previous section, the following property holds.

$$X' \geq 0 \quad \Leftrightarrow \quad X < p_0 p/2 \tag{39}$$

$$\Leftrightarrow \quad [X/p] < p_0/2. \tag{40}$$

Let $Y = Y_1 q_1 + Y_2 q_2 + \cdots + Y_m q_m$ where $q_i = p/p_i$, $u_i q_i \equiv 1 \bmod p_i$, $Y_i = (X_i u_i) \% p_i$. As we have already had in previous section,

$$[X/p] \equiv c(X_0 - Y) + [Y/p] \bmod p_0 \tag{41}$$

where c is an integer satisfying $cp \equiv 1 \bmod p_0$, which can be computed in the preprocess.

From the definition of Y, we have

$$\frac{Y}{p} = \frac{Y_1}{p_1} + \frac{Y_2}{p_2} + \cdots + \frac{Y_m}{p_m} \ (< m). \tag{42}$$

Since $2^n - 2^{n/2} < p_i < 2^n$ $(i = 1, \cdots, m)$, from this equation we have following inequalities by replacing p_i with $2^n - 2^{n/2}$ and 2^n:

$$\frac{Y_1 + \cdots + Y_m}{2^n} \leq \frac{Y}{p} \leq \frac{Y_1 + \cdots + Y_m}{2^n - 2^{n/2}}. \tag{43}$$

We take the most left side as an approximation of Y/p. Since $Y_i < p_i < 2^n$, $m \leq 2^{n/2} - 1$, we estimate the difference between this approximation and Y/p as follows:

$$0 \leq \frac{Y}{p} - \frac{Y_1 + \cdots + Y_m}{2^n} \leq \frac{Y_1 + \cdots + Y_m}{2^n - 2^{n/2}} - \frac{Y_1 + \cdots + Y_m}{2^n} \tag{44}$$

$$= \frac{2^{n/2}(Y_1 + \cdots + Y_m)}{(2^n - 2^{n/2})(2^n)} \tag{45}$$

$$< \frac{2^{n/2} m 2^n}{(2^n - 2^{n/2})(2^n)} \leq \frac{2^{n/2}(2^{n/2} - 1)}{(2^n - 2^{n/2})} = 1 \tag{46}$$

This means the the difference of their integral parts is at most 1:

$$[(Y_1 + \cdots + Y_m)/2^n] \leq [Y/p] \leq [(Y_1 + \cdots + Y_m)/2^n] + 1. \tag{47}$$

We can compute $[(Y_1 + \cdots + Y_m)/2^n]$ by the following procedure: First, set a counter $Z \leftarrow 0$, then, add $Y_2, Y_3, \cdots, Y_m$ to Y_1 one by one (the sum is computed in lower n bit) and in every time the sum becomes more than $n + 1$ bit, increment the counter Z by one.

After this procedure, counter Z shows $[(Y_1 + \cdots + Y_m)/2^n]$. Since $[Y/p] - 1 \leq Z \leq [Y/p]$,

$$c(X_0 - Y) + Z \leq c(X_0 - Y) + [Y/p] \leq c(X_0 - Y) + Z + 1 \tag{48}$$

From this inequalities and the formula 41, we have the following properties.

$$(c(X_0 - Y) + Z) \% p_0 < p_0/2 \quad \Rightarrow \quad [X/p] < p_0/2, \tag{49}$$

$$(c(X_0 - Y) + Z + 1) \% p_0 \geq p_0/2 \quad \Rightarrow \quad [X/p] \geq p_0/2. \tag{50}$$

In these 2 cases, we obtain the sign of X'. In the other case (i.e. $(c(X_0 - Y) + Z) \% p_0 \geq p_0/2$ and $(c(X_0 - Y) + Z + 1) \% p_0 < p_0/2$), we can easily see $(c(X_0 - Y) + Z) \% p_0 = p_0 - 1$. This means

$$[X/p] = p_0 - 1 \text{ or } 0, \quad \text{i.e. } -p \leq X' < p \tag{51}$$

In this case, we can apply this method to $(p_0 - 1)X'$ instead of X' since $-p_0 p/2 \leq (p_0 - 1)X' < p_0 p/2 - p$. If necessary, we can iterate it. The iteration terminates at most $m + 1$ times

Time complexity is $O(m^2)$ and space one is $O(m)$.

7 Concluding Remarks

We introduce 4 methods to determine the sign of an integer of m-ple length from its reminders. Time complexities are $O(m^2)$ and space ones are $O(m^2)$ and $O(m)$.

In theoretical sense, the method to reconstruct a m-ple integer directly from its remainders in $O(m(\log m)^2 \log\log m)$[1, 5]. This is theoretically faster than all methods presented in this paper but it is practically too slow for geometric algorithms.

If a geometric algorithm requires a sign-test of an integer of m-ple length while all input data are within the unit length, the algorithm usually contains $O(m)$ multiplications or $O(m^2)$ additions/subtractions before the sign-test. The algorithm takes at least $O(m^2)$ time as long as all variables in the algorithm has the same fixed bits. So we can say in practical sense, there is little need to do the sign-test in less than $O(m^2)$ time.

The methods presented in this paper is simple and practically fast, so we come to use modular arithmetics in impementing geometric algorithms.

References

[1] Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[2] Edelsbrunner, H. and Mücke, E. P.: Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms, *Proc. 4th ACM Ann. Symp. on Computational Geometry*, pp. 118–133, 1988.

[3] Emiris, I. and Canny, J.: An Efficient Approach to Removing Geometric Degeneracies, *Proc. 8th Ann. Symp. on Computational Geometry*, pp. 74–82, 1992.

[4] Fortune, S.: Polyhedral Modelling with Exact Arithmetic, *Proc. 3rd IEEE Symp. Solid Modelling and Applications*, 1995.

[5] Knuth, D. E.: *The Art of Computer Programming*, Vol. 2, *Seminumerical Algorithms*, 2nd ed., Addison-Wesley, 1981.

A quadratic non-standard arithmetic

Dominique Michelucci
École des Mines, F-42023 Saint-Étienne 02
micheluc@emse.fr

1 Introduction

This extended abstract presents an exact real quadratic arithmetic which also handles infinitely small numbers. The quadratic arithmetic provides the same operations than an exact rational one, plus square root of non negative numbers. It computes in the real quadratic closure of $\mathbb{Q}$, noted here: $\bar{\mathbb{Q}}$. As for an example, such an arithmetic can be used to compute the $2D$ arrangement of a set of circles and lines, or the $3D$ arrangement of a set of spheres and planes. The quadratic arithmetic is classic and presented in section 2. The new part is the way infinitely small numbers are managed in the quadratic framework.

In Computational Geometry, infinitely small numbers are typically used to symbolically perturb input parameters [2, 11, 10, 3, 7]: this infinitesimal perturbation removes accidental dependencies between data, and thus eliminates geometric degeneracy (alignment of more than two points, cocircularity of more than three points, etc) so that programmers have to handle only the few generic cases. The more often, people put forward efficiency arguments to restrict used perturbation schemes: most of the time, the perturbation is valid only for a single geometric predicate (say the InCircle predicate), and infinitesimals are not explicitly represented inside the computer. As a consequence, geometric predicates must concern input parameters only, and not derived values; moreover the user has poor control, if any, on the perturbation, typically seen as a black box. In opposition, the arithmetic in this extended abstract enables the user to compute with infinitesimals like if they were ordinary numbers in $\bar{\mathbb{Q}}$. Thus this arithmetic can manage in a straightforward way all proposed perturbation schemes (assuming the quadratic framework is sufficient, of course). Here are some of them:

- In Yap's perturbation [11, 10, 7], each input parameter x_i is perturbed into $x_i + \epsilon_i$. Thus $f(x + \epsilon) = \sum_\alpha \frac{1}{\alpha!}\epsilon^\alpha f^{(\alpha)}(x)$ by Taylor's formula. A consistent ordering between all power products $\epsilon^\alpha = \epsilon_1^{\alpha_1}\epsilon_2^{\alpha_2}\ldots$ sorts derivatives in decreasing magnitude: the sign of $f(x+\epsilon)$ is the one of the first non vanishing term in that sequence. Actually, ϵs are only implicit in Yap's perturbation.

- Each input parameter x_i is perturbed into $x_i + g_i\epsilon$, where g_i describe a generic situation, in Emiris and Canny's work [3]. Then $x_i + g_i\epsilon$ is a generic configuration: a single ϵ is sufficient.

- Each input parameter x_i is perturbed into $x_i + a_i\epsilon$, where a_i is random. The idea is that random a_i are very probably generic [3]. That is a probabilist perturbation scheme.

- See quoted references for other perturbation schemes.

This quadratic arithmetic cannot guarantee that the used perturbation scheme is valid, *ie* really removes all degeneracy: that obviously remains the responsibility of the CGer. But it detects persistent degeneracy at run time.

2 A Quadratic Arithmetic

2.1 Towers of extensions

The material in this section is not new [6], it is here only for completeness. In Fortune's method [4], one has to compare numbers of the form $\frac{a+\sqrt{b}}{c}$, where a, b, c are integers. It is possible to use repeated squaring, for this restricted case. This section presents a more general *real quadratic arithmetic*, which provides exact comparisons and operations: $+$, $-$, $\div$, $\times$ and $\sqrt{}$ on non negative numbers, starting from $\mathbb{Q}$. This arithmetic is an alternative to Yap and Dubé's one [1, 9]. Lack of space does not permit a comparison.

The idea is to compute in a tower of Real quadratic extensions $K_0 = \mathbb{Q}, \ldots K_i = K_{i-1}(\sqrt{\alpha_{i-1}})$ where K_i is an algebraic (and quadratic) extension over K_{i-1}, and $\alpha_{i-1} \in K_{i-1}$ is Real and positive and has no square roots in K_{i-1}. It means $K_i = K_{i-1}(\sqrt{\alpha_{i-1}})$ is the set of the numbers $u + v\sqrt{\alpha_{i-1}}$, with u and v two elements in K_{i-1}: in other words, numbers in K_i are represented by a vector of two components: u, $v \in K_{i-1}$, and K_i is represented by $\alpha_{i-1} \in K_{i-1}$ (which we already know how to represent, by induction) and by some reference to K_{i-1}. Operations in K_i straightforwardly reduce to operations in K_{i-1}:

$$
\begin{aligned}
&(u + v\sqrt{\alpha_{i-1}}) + (u' + v'\sqrt{\alpha_{i-1}}) = (u + u') + (v + v')\sqrt{\alpha_{i-1}} \\
&(u + v\sqrt{\alpha_{i-1}}) \times (u' + v' \times \sqrt{\alpha_{i-1}}) = (u \times u' + v \times v' \times \alpha_{i-1}) + (u \times v' + u' \times v)\sqrt{\alpha_{i-1}} \\
&-(u + v\sqrt{\alpha_{i-1}}) = (-u) + (-v)\sqrt{\alpha_{i-1}} \\
&1/(u + v\sqrt{\alpha_{i-1}}) = (u/[u^2 - \alpha_{i-1} \times v^2]) - (v/[u^2 - \alpha_{i-1} \times v^2])\sqrt{\alpha_{i-1}}
\end{aligned}
$$

Computing the sign of $w = u + v\sqrt{\alpha_{i-1}} \in K_i$ also boils down to computations in K_{i-1}:

$$
\begin{aligned}
&u = 0 \text{ or } v = 0 : \text{ trivial} \\
&u > 0 \text{ and } v \geq 0 \Rightarrow w > 0 \\
&u > 0 \text{ and } v < 0 \Rightarrow \text{sign}(w) = \text{sign}(u^2 - v^2\alpha_{i-1}) \\
&u < 0 \Rightarrow \text{sign}(w) = -\text{sign}(-w)
\end{aligned}
$$

and in the end $K_0 = \mathbb{Q}$, where we know how to compute a sign, so the recursion eventually stops.

The last required operation is the square root in K_i. Assume $w = u + v\sqrt{\alpha_{i-1}} \in K_i$ is positive. The first thing is to test if w is a square in K_i, say the square of $z \in K_i$ with $z = x + y\sqrt{\alpha_{i-1}} > 0$ with $x, y \in K_{i-1}$. We suppose u and v do not vanish, because this case trivially reduces to the same problem in K_{i-1}.

$$
\begin{aligned}
&w = u + v\sqrt{\alpha_{i-1}} = (x + y\sqrt{\alpha_{i-1}})^2 \\
&\Leftrightarrow u = x^2 + \alpha_{i-1} \times v^2 \text{ and } v = 2 \times x \times y \\
&\Leftrightarrow x^2 = \frac{1}{2}\left[u \pm \sqrt{u^2 - \alpha_{i-1} \times v^2}\right] \text{ and } v = 2 \times x \times y
\end{aligned}
$$

Thus $w \in K_i$ is a square in K_i iff $u^2 - \alpha_{i-1} \times v^2$ is a square in K_{i-1} and if $\frac{1}{2}\left[u + \sqrt{u^2 - \alpha_{i-1} \times v^2}\right]$ or $\frac{1}{2}\left[u - \sqrt{u^2 - \alpha_{i-1} \times v^2}\right]$ is a square in K_{i-1} (Note that they cannot be both squares in K_{i-1} because their product: $\frac{\alpha_{i-1}v^2}{4}$ is not a square in K_{i-1}).

Thus testing if $w \in K_i$ is a square in K_i reduces to computations in K_{i-1}: in the end, testing if $w \in K_0 = \mathbb{Q}$ is a square in $\mathbb{Q}$ is trivial. If w is a square in K_i, the method also gives its positive square root $x + y\sqrt{\alpha_{i-1}}$. When $w \in K_i$ is not a square in K_i, we have to define the quadratic extension of K_i which contains the square root of w: call this extension $K_{i+1} = K_i(\sqrt{w})$. In particular, the coordinates of $\sqrt{w}$ in K_{i+1} are: $(0 \in K_i, 1 \in K_i)$.

2.2 Using dags

Using a single tower of quadratic extensions to perform all computations results in very poor performance, since the first required operation is to express all met numbers (even integers or rationals) in the higher field. Thus the time required for a new computation increases with the number of previously performed computations, even when they are independent. It is clearly unacceptable. A possible solution is as follows.

Numbers are first represented by expressions, or dags (Directed Acyclic Graphs), like in the lazy rational arithmetic [8] or Dubé and Yap's arithmetic. A dag may be: a usual rational number, the sum or the product of two other dags, the opposite, the reciprocal or the square root of another dag. Each dag is associated with an enclosing interval, computed with arithmetic interval. When the interval becomes insufficient, the dag at hand is evaluated in an exact way, *starting from an empty tower,* ie *a tower containing only* $\mathbb{Q}$. This way, the time needed to exactly evaluate a dag depends only on the complexity of the dag itself (and of its subdags of course, but not on previously evaluated dags). Moreover, only inevitable exact computations are performed.

This optimization exploits the fact that CG typically deals with a lot of little computation trees, instead of a big one or a few big ones like in Symbolic Computing. Actually, the depth of computation trees met in classical non re-entrant CG methods (Convex Hulls, say) can even be known a priori: it is a constant for CG methods working in 2D or 3D, and a linear function of the underlying space dimension for CG methods working in $\mathbb{R}^n$ (actually $\mathbb{Q}^n$ or $\bar{\mathbb{Q}}^n$).

2.3 Computing characteristic polynomials

If need be, the characteristic polynomial of $z \in \bar{\mathbb{Q}}$ can be computed by the following method. Assume the characteristic polynomial of z is known in some extension $K(\alpha = \sqrt{A})$: it is $f(z) = \sum_{i=0}^{d} f_i z^i = 0$, with $f_i = a_i + b_i\alpha$, where a_i and b_i are in K. Then the equation of z in K is obtained with:

$$g(z) = \left[\sum_{i=0}^{d} a_i z^i\right]^2 - A\left[\sum_{i=0}^{d} b_i z^i\right]^2 = 0$$

and has degree twice f degree. Now an initial characteristic polynomial of z is easily computed in the highest extension of the tower: the polynomial of $z = x + y\sqrt{A}$ is $(z-x)^2 - y^2A = 0$.

Another possible method gives a companion matrix, the characteristic polynomial of which is the one of z. Then interpolation methods (e.g. Lagrange) can be used to recover the characteristic polynomial of the matrix. Let $z = x + y\alpha \in K(\alpha = \sqrt{A})$, with $x, y \in K$. Multiplication of any number $a + b\alpha \in K(\alpha)$ and z can be seen as a linear transformation of vector $(a\ b)$ by the companion matrix:

$$M(z) = \begin{pmatrix} x & y \\ yA & x \end{pmatrix}$$

More generally, each number $z \in K(\alpha)$ can be represented by $M(z)$. In particular, the characteristic equation of $M(z)$ is the one of z. Recursive replacements of $M(z)$ entries by other matrices eventually reaches a matrix with rational coefficients. For instance, let $z = x+y\sqrt{3}$ with $x = a+b\sqrt{2}$ and $y = c+d\sqrt{2}$. Then z is represented by companion matrix

$$M(z) = \begin{pmatrix} x & y \\ 3y & x \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} a & b \\ 2b & a \end{pmatrix} & \begin{pmatrix} c & d \\ 2d & c \end{pmatrix} \\ 3\begin{pmatrix} c & d \\ 2d & c \end{pmatrix} & \begin{pmatrix} a & b \\ 2b & a \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ 2b & a & 2d & c \\ 3c & 3d & a & b \\ 6d & 3c & 2b & a \end{pmatrix}$$

3 Computing with infinitely small numbers

Let $\mathbb{G}$ be a *computable* real quadratic field. Initially $\mathbb{G} = \bar{\mathbb{Q}}$, which the previous section has shown to be computable, *ie* we can perform additions, multiplications, opposites, reciprocals, comparisons, square roots of non negative numbers in $\mathbb{G}$. We introduce a new and infinitely small number, namely ϵ, which is smaller than all positive numbers in $\mathbb{G}$. This section shows that $\mathbb{G}(\epsilon)$ is also a *computable* real quadratic field. Then $\mathbb{G}(\epsilon)$ can itself be used as a new background field for another non standard extension, with another new infinitely small number, namely ϵ', which is smaller than all positive numbers in $\mathbb{G}(\epsilon)$. This way it is possible to compute in a tower of non standard extensions: that may be useful when the CG problem at hand does not admit a perturbation scheme with a single ϵ. Recall if $g = (g_1, g_2 \ldots g_n)$ is a generic input of the same problem, whose input parameters are: $p = (p_1, p_2 \ldots p_n)$, then $p + \epsilon g$ is generic; but it may be a difficult realizability question to find such a generic configuration for some problem, like for instance finding a polytope with a prescribed topology.

3.1 The non quadratic case

In the simplest case, square roots and divisions are not used. Numbers reachable from $\mathbb{G}$ and ϵ are thus polynomials in ϵ: $P(\epsilon) = a_0 + a_1\epsilon + \ldots + a_n\epsilon^n$, where $a_i \in \mathbb{G}$. The sign of $P(\epsilon)$ is clearly the coefficient sign of its first non vanishing term. The naive method computes all coefficients of polynomials, using standard symbolic computations. A more clever method represents polynomials with (not inevitably minimal) dags, in the now usual way, and exploits laziness and Taylor's formula:

$$P(\epsilon) = P(0) + \epsilon P'(0) + \frac{\epsilon^2}{2!}P''(0) + \ldots \frac{\epsilon^d}{d!}P^{(d)}(0)$$

The sign of $P(\epsilon)$ is the one of the first non vanishing term in the sequence: $P(0)$, $P'(0)$, $P''(0) \ldots P^{(d)}(0)$, where d is P degree (or an upper bound). When all coefficients vanish, $P(\epsilon)$ is identically zero and the sign is zero. When polynomial P is represented by some dag, it is easy to generate on-the-fly a dag for its derivative P' (and then for P'', P''', etc), using standard derivation rules. Successive derivations of a dag eventually reach a dag without occurrence of ϵ, possibly at the cost of some supplementary derivation. For instance the dag: $d_0(\epsilon) = (\epsilon - \epsilon)$ is a non minimal dag representing the null polynomial. It vanishes in $\epsilon = 0$; it has derivative: $d_1 = (1 - 1)$ which also vanishes in 0. Since d_1 contains no more occurrence of ϵ, d_1 and d_0 are proven to be identically zero.

To summarize: if a dag $f(\epsilon)$ does not contain ϵ, its sign is trivially computed in $\mathbb{G}$. Otherwise, when $f(0)$ does not vanish, its sign gives the sign of $f(\epsilon)$. Otherwise the sign of $f(\epsilon)$ is the one of $f'(\epsilon)$. This process always ends. Note dags are evaluated in $\epsilon = 0$, thus no really symbolic computations (like polynomial gcd or resultants) are performed on polynomials, except dag derivation and construction.

This method can be seen as an extension of Yap's perturbation, whose limitations are overcome: the user has full control on the perturbation (recall Yap's perturbation systematically and implicitly perturbs the ith input parameter p_i into $p_i + \epsilon_i$): it is easy to choose the sign of the perturbation or to force a perturbed point to stay on a given half straight line for instance. Moreover the programmer is relieved of the burden of expressing all tests with input parameters only: dag handling does the job. It is true that towers of non standard extensions: $\left(\bar{\mathbb{Q}}(\epsilon)\right)(\epsilon')$... implicitly sort the set of power products in $\epsilon, \epsilon' \ldots$ with reverse lexicographic order, *ie* :

$$1 >> \epsilon >> \epsilon^2 >> \epsilon^3 \ldots >> \epsilon' >> \epsilon\epsilon' >> \epsilon^2\epsilon' >> \epsilon^3\epsilon' \ldots >> \epsilon'^2 >> \epsilon\epsilon'^2 >> \epsilon^2\epsilon'^2 >> \epsilon^3\epsilon'^2 \ldots > 0$$

though the user may prefer another *consistent ordering* (also called: *compatible ordering*), say for instance total degree ordering. But the user can easily obtain any other compatible order [7], for instance using: $\alpha = \epsilon'$ and $\alpha' = \epsilon\epsilon'$, he gets α and α' power products sorted in total then reverse lexicographic order, *ie* :

$$1 >> \alpha >> \alpha' >> \alpha^2 >> \alpha\alpha' >> \alpha'^2 >> \alpha^3 >> \alpha^2\alpha' >> \alpha\alpha'^2 >> \alpha'^3 \ldots > 0$$

3.2 Quadratic case

Unfortunately, the previous solution doesn't extend with divisions and square roots: some derivatives can be undefined in 0, like $F(\epsilon) = \sqrt{\epsilon}$ the derivative of which is $F'(\epsilon) = \frac{1}{2\sqrt{\epsilon}}$. Moreover, successive derivations may never reach an expression without occurrence of ϵ. The proposed solution uses series in ϵ.

3.2.1 Using series in ϵ

Each element of $\mathbb{G}(\epsilon)$ is first represented by a dag: a dag can be a constant in $\mathbb{G}$, ϵ, the sum or the product of two other dags, the opposite or the reciprocal or the square root of another dag. Moreover, each dag is associated with a series $S(\epsilon)$, represented by a *factor* $k \in \mathbb{N}$, an *algebraic degree* $d \in \mathbb{N}$ (defined below), a shift power $p \in \mathbb{N}$, and the lazy (potentially infinite) list or array a_i of its coefficients in $\mathbb{G}$:

$$\delta = \epsilon^{(\frac{1}{2^k})} \ , \ S(\epsilon) = \frac{a_0 + a_1\delta + a_2\delta^2 + \ldots}{\delta^p}$$

The dag makes possible the computation of coefficient a_i when needed. The factor k is needed because of expressions like $\sqrt{\epsilon}$. It is easy to convert a series to another one with a greater factor, thus we can suppose w.l.o.g. that series to be added or multiplied have the same factor. The shift power p typically equals 0, it is only used to avoid negative exponents, like in $\frac{1}{\epsilon} = \epsilon^{-1}$. It is ignored from now on. We give formulas for coefficients of sum, product, inverse and square root:

$(a_0 + a_1\delta + \ldots) + (b_0 + b_1\delta + \ldots) = x_0 + x_1\delta + \ldots$ where $x_k = a_k + b_k$

$(a_0 + a_1\delta + \ldots)(b_0 + b_1\delta + \ldots) = x_0 + x_1\delta + \ldots$ where $x_k = \sum_{i=0}^{i=k} a_i b_{k-i}$

$\frac{1}{a_0+a_1\delta+\ldots} = x_0 + x_1\delta + \ldots$ where $x_0 = \frac{1}{a_0}$, $x_k = -\frac{1}{a_0}\sum_{i=1}^{k} a_i x_{k-i}$, assuming $a_0 \neq 0$

$\sqrt{a_0 + a_1\delta + \ldots} = x_0 + x_1\delta + \ldots$ where $x_0 = \sqrt{a_0}$, $x_k = \frac{1}{2x_0}\left(a_k - \sum_{i=1}^{i=k-1} x_i x_{k-i}\right)$ assuming $a_0 > 0$.

If $a_0 < 0$ there is no square root. If $a_0 = 0$, let $\delta_2 = \sqrt{\delta}$, then $\sqrt{S(\epsilon)} = \delta_2\sqrt{a_1 + a_2\delta_2^2 + a_3\delta_2^4 \ldots}$

3.2.2 A gap theorem for sign computation

The sign of a non identically null series is the sign of its first non vanishing coefficient. When the perturbation scheme really removes all degeneracy, series identically null cannot occur. However their detection is useful. It is made with the following *gap theorem*: a series $y = S(\epsilon)$ is identically zero when all its coefficients in terms $a_i\epsilon^i$ with $i \leq d$ are zero, where d is the *algebraic degree*, or an upper bound, of the series.

Let $y = f(\epsilon)$, where $f(\epsilon)$ is a dag in the variable ϵ, and let $S(\epsilon)$ be the corresponding series. ϵ and y fulfill a polynomial condition: $F(\epsilon, y) = 0$. In other words, the point (ϵ, y) lies on an algebraic curve. The degree of the dag f, and of the corresponding series, is the total degree of the polynomial F in ϵ and y: for instance, if $y = f(\epsilon) = \sqrt{1+\epsilon}$, then $F(\epsilon, y) = y^2 - (1+\epsilon) = 0$ has degree 2. Same degree for $y = f(\epsilon) = (1+\epsilon)^2$. d, the degree or an upper bound, is recursively computed like that: if the dag is ϵ or a constant in $\mathbb{G}$, then its degree is 1. If the dag is the sum or product of two other dags a and b, its degree is the product of the degrees for a and b. The degree of $-a$ and $1/a$ is the degree of dag a. The degree for $\sqrt{a}$ is twice the degree of dag a.

We sketch now the proof of the gap theorem. Assume for simplicity that the factor of the series is 0, *ie* the series is $y = S(\epsilon) = a_0 + a_1\epsilon + a_2\epsilon^2 + \ldots$. The coefficients for negative exponents are zero (otherwise, the series is clearly not 0!). The degree of the dag $y = f(\epsilon)$ is at most d. Now, $a_0 = a_1 \ldots = a_d = 0$. Thus the curve branch $y = s(\epsilon) = a_0 + a_1\epsilon + a_2\epsilon^2 + \ldots$ cuts the straight line $y = 0$ at the origin, with multiplicity $d+1$. But this branch is part of an algebraic curve $F(\epsilon, y) = 0$ with degree not greater than d. Thus the curve branch $y = f(\epsilon)$ is the straight line $y = 0$, and the series $S(\epsilon)$ is identically 0.

This argument extends as follows when the series factor is greater than 0. The dag $y = f(\epsilon)$ has always algebraic degree d but the corresponding series is now: $S(\epsilon) = a_0 + a_1[\epsilon^{\frac{1}{2^k}}]^1 + a_2[\epsilon^{\frac{1}{2^k}}]^2 + \ldots$. Let $\delta = \epsilon^{\frac{1}{2^k}} \Leftrightarrow \epsilon = \delta^{(2^k)}$ and $T(\delta) = a_0 + a_1\delta^1 + a_2\delta^2 + \ldots$. The dag $y = g(\delta) = f(\delta^{(2^k)})$ has algebraic degree $d \times 2^k$. From the previous gap theorem, when all coefficients for exponents $0, 1 \ldots, d \times 2^k$ of $T(\delta)$ vanish, then the series $y = T(\delta) = S(\epsilon)$ is identically zero. Conclude.

3.3 Limitations of perturbation methods

Though input parameters are perturbed according some perturbation scheme, computations may still yield to non generic situations. It is especially true, and annoying, with on-line or reentrant methods, where computed values may be re-used (for instance an intersection point becomes a vertex of a new edge). The most trivial example is as follows: the intersection point I between two perturbed segments AB and CD (whatever the perturbation) is exactly aligned with A and B, and with C and D! Alignment of more than 2 points is a degeneracy[1]. Note geometric applications in the real world involve edition of geometric objects, thus they need reentrant methods. These limitations of perturbation have not been seriously addressed so far.

4 Conclusion

This extended abstract has presented a quadratic arithmetic which also handles infinitely small numbers, like if they were ordinary numbers. The computation depth has not to be known or bounded *a priori* (contrarily to other CG exact arithmetics, like Fortune and Van Wyk's one [5]). Perturbations are handled using lazy series and a gap theorem to achieve termination. Exploiting laziness insures that only inevitable computations are performed. This arithmetic can be used with all existing perturbation schemes (compatible with a quadratic arithmetic). It detects at run time cases in which the perturbation scheme does not remove degeneracy. It does not add any overhead (except constant) to the intrinsic complexity of asked computations. Unfortunately, due to lack of space, this extended abstract couldn't compare it with other quadratic or algebraic arithmetics.

References

[1] T. Dubé and C. Yap. A basis for implementing exact geometric algorithms. *manuscript*, 1993.

[2] H. Edelsbrunner and E.P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph*, 9:66–104, 1990.

[3] I. Emiris and J. Canny. A general approach to removing degeneracies. *SIAM J. Computing*, 24(3):650–664, June 1995.

[4] S. Fortune. A sweep-line algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[5] S. Fortune and C. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proceedings of the 9th ACM Symposium on Computational Geometry*, pages 163–172, San Diego, May 1993.

[6] D. Michelucci. The robustness issue. *submitted to publication*, 1997.

[7] D. Michelucci. An epsilon-arithmetic for removing degeneracies. In *Proceedings of the IEEE 12th Symposium on Computer Arithmetic*, pages 230–237, Windsor, Ontario, July 1995.

[8] D. Michelucci and J-M. Moreau. Lazy arithmetic. *To be published in IEEE Transactions on Computers*, summer 1997. Available at: <ftp://ftp.emse.fr/pub/papers/LAZY/lazy.ps.gz>.

[9] K. Ouchi. Implementation of exact computation. *Masters Thesis, New York University*, 1997.

[10] C.K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *J. Comput. Syst. Sci.*, 40:2–18, 1990.

[11] C.K. Yap. Symbolic treatment of geometric degenaracies. *J. Symbolic Comput*, 10:349–370, 1990.

[1] A solution is here to perturb the intersection point with a new infinitesimal smaller than all already existing ones

Analysis of a class of k-dimensional merge procedures, with an application to $2D$ Delaunay Triangulation in expected linear time after two-directional sorting

Christophe Lemaire[1,2] Jean-Michel Moreau[2]

Extended abstract

[1] SETRA, 46 AVENUE ARISTIDE BRIAND, BP 100, 92223 BAGNEUX CEDEX, FRANCE (lemaire@setra.fr)
[2] E.M.S.E., 158 COURS FAURIEL, 42023 SAINT-ÉTIENNE, FRANCE (moreau@emse.fr)

Abstract

This paper exploits the notion of "unfinished sites" in the average-case analysis of k-dimensional divide-and-conquer algorithms.

This general result is then applied to the $2D$ case, and it is shown that the divide-and-conquer construction of the Delaunay triangulation of a set of planar points quasi-uniformly distributed in a square may be done in expected linear time after a two-directional preprocessing sort. This method is readily implemented, and, as shown by the easily reproduced results provided, it is one of the fastest worst-case optimal methods ever suggested to construct real-scale Delaunay triangulations in the plane.

1 Introduction

The first worst-case optimal, $\Theta(n \log n)$, method to construct the Delaunay triangulation in the plane was published in 1980 by Lee and Schachter ([8]). Their method – which will be referred to in the text as "the standard algorithm" – was based on the famous divide-and-conquer paradigm. In 1985, Guibas and Stolfi generalized this method ([6]). Meanwhile, in 1984, A. Maus ([9]) published the first expected linear-time method for constructing the Delaunay triangulation of a site of planar points, using a grid to speed up the location of the closest empty-circle neighbour for any given Delaunay edge.

In 1987, R. Dwyer published an $O(n \log \log n)$ method ([4]) for the same problem assuming uniform distribution, using a regular grid. In their 1988 paper, [7], J. Katajainen and M. Koppinen introduced a variant of the standard algorithm, based on a regular grid. Their average-case analysis exploited the notion of unfinished sites in a rectangular domain. Finally, R. Dwyer published the first k-dimensional method with expected linear behaviour in 1991 ([5]).

This paper addresses the analysis of k-dimensional divide-and-conquer methods that make use of the notion of unfinished site and of (balanced) kd-tree based partitioning. Some preliminary definitions and mathematical results are given in Section 2. Section 3 then presents the main result for dimension k, that is exploited, in Section 4, to obtain the construction of the Delaunay triangulation of a set of planar points quasi-uniformly distributed in a square, in expected linear time, after a two-directional sort preprocessing step.

2 Definitions and mathematical preliminaries

We first need to prepare the ground for the general analysis of Section 3 by presenting its framework and by giving some preliminary definitions or results that will be required afterwards.

Although the analysis has a wider scope, let us, for the time being, concentrate on the k-dimensional Delaunay triangulation, noted $DT(S)$, of a set S of *sites*. We shall use the term "triangulation" for k-triangulation ($k \geq 2$) throughout this paper.

Assume the sites to be quasi-uniformly distributed on a unit cube $\mathcal{U}_k$. This implies the existence of two strictly positive real constants $c_1 \leq c_2$, and of a probability density f such that,

$$\forall X = (x_1, x_2, \ldots, x_k) \in \mathcal{U}_k, c_1 \leq f(X) \leq c_2,$$

and f is null outside the cube. Thus, the probability for one given site to lie in domain $\mathcal{D}$ is $\int_{\mathcal{D}} f$.

2.1 Unfinished sites

Let us generalize some definitions and statements from [7]:

Definition 2.1 *Let $T(S_1)$ and $T(S_2)$ be the triangulations of sets S_1 and S_2. We shall write $T(S_1) <_\Delta T(S_2)$ whenever $S_1 \subseteq S_2$ and $T(S_1)$ contains each edge in $T(S_2)$ the endpoints of which belong to S_1.*

Definition 2.2 *Let $T(S_1) <_\Delta T(S_2)$ be two triangulations and $s \in S_1$. We say that site s is* finished *in*

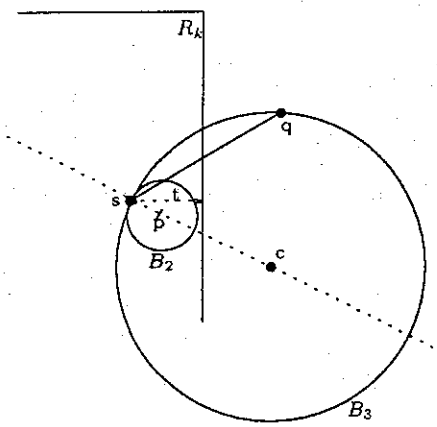


Figure 1: *Lemma 2.1.*

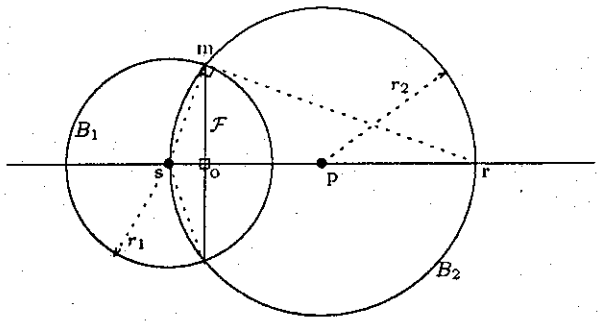


Figure 2: *Lemma 2.2.*

$T(S_1)$ wrt $T(S_2)$ if the set of edges adjacent to s in $T(S_1)$ and $T(S_2)$ coincide; otherwise, s is said to be unfinished *(in $T(S_1)$ wrt $T(S_2)$).*

The first definition induces a partial order in the set of triangulations. Notice that $T(S_1) <_\Delta T(S_2)$ does not imply that all edges in $T(S_1)$ belong to $T(S_2)$. Also note that $DT(S_1) <_\Delta DT(S_2)$ is equivalent to $S_1 \subseteq S_2$ if S_1 has a unique Delaunay triangulation.

The two following results are direct consequences of Definition 2.2:

Proposition 2.1 *Let $T(S_1) <_\Delta T(S_2)$ be two triangulations. A site $s \in S_1$ is finished in $T(S_1)$ wrt $T(S_2)$ if and only if S_1 contains the endpoints of all edges adjacent to s in $T(S_2)$.*

2.2 k-partition around unfinished sites

Lemma 2.1 *If s is an unfinished site in k-rectangle $\mathcal{R}_k$, at distance t from the boundary of $\mathcal{R}_k$, then there is a k-ball $\mathcal{B}_2$ with radius $\frac{t}{2}$, centered at distance $\frac{t}{2}$ from s, with no sites in its interior.*

Proof: Referring to Figure 1, if s is an unfinished site in k-rectangle $\mathcal{R}_k$, there is a site q belonging to $\mathcal{U}_k \backslash \mathcal{R}_k$ such that (s,q) is a Delaunay edge in $\mathcal{U}_k$. Since q lies outside k-rectangle $\mathcal{R}_k$, the length of (s,q) is greater than t. And since (s,q) is a Delaunay edge, we may find a k-ball $\mathcal{B}_3$ with no sites in its interior, and s and q on its boundary. The diameter of this k-ball is thus greater than t. Let c be the center of $\mathcal{B}_3$, and $\mathcal{B}_2$ the k-ball with center lying on segment $[sc]$ at distance $\frac{t}{2}$ from s. $\mathcal{B}_2$ is contained in $\mathcal{B}_3$, hence $\mathcal{B}_2$ has no sites in its interior.

Lemma 2.2 *Let s be a site, and $\mathcal{B}_1$ the ball centered in s, with radius r_1. Let $\mathcal{B}_2$ be the ball with radius $r_2 = \frac{t}{2}$, and center p at distance $\frac{t}{2}$ from s. Let $(s, \vec{\imath}_1, \ldots, \vec{\imath}_k)$ be an orthonormal system of coordinates, with $\vec{\imath}_1$ collinear to $\overrightarrow{sp}$ and with same direction. Let $\mathcal{R}o$ be the portion of 'rectangular' cone with apex s, with symmetry axis $\vec{\imath}_1$, included in $\mathcal{B}_1$, and with equation in a generalized spherical coordinate system $(\rho, \alpha_1, \ldots, \alpha_{k-1})$:*

$$(\mathcal{R}o) \begin{cases} \rho & < & r_1 & & \\ -\frac{\pi}{4} & < & \alpha_1 & < & \frac{\pi}{4} \\ \vdots & & \vdots & & \vdots \\ -\frac{\pi}{4} & < & \alpha_{k-1} & < & \frac{\pi}{4} \end{cases}$$

then $\quad \mathcal{R}o \subset \mathcal{B}_2 \Leftrightarrow r_1 \leq \frac{t}{2^{\frac{k-1}{2}}}$

Proof: Let us first show that $\mathcal{R}o \subset \mathcal{B}_2 \Rightarrow r_1 \leq \frac{t}{2^{\frac{k-1}{2}}}$.

Knowing that $\mathcal{R}o \subset \mathcal{B}_2$ implies that, for all $\alpha_i = \pm\frac{\pi}{4}$, $m(r_1, \alpha_1, \ldots \alpha_{k-1})$ belong to $\mathcal{B}_2$.

Now, consider the limiting case where points m are situated on the common boundary of balls $\mathcal{B}_1$ and $\mathcal{B}_2$ (Figure 2). We have : $so = \frac{r_1}{(\sqrt{2})^{k-1}}$

Since triangles $\triangle som$ and $\triangle smr$ are similar:

$$\frac{so}{sm} = \frac{sm}{sr} \Rightarrow \frac{1}{(\sqrt{2})^{k-1}} = \frac{r_1}{t} \Rightarrow r_1 = \frac{t}{2^{\frac{k-1}{2}}}$$

whence: $\mathcal{R}o \subset \mathcal{B}_2 \Rightarrow r_1 \leq \frac{t}{2^{\frac{k-1}{2}}}$

The reciprocal is straightforward.

Corollary 2.1 *Consider te $(k-1)$-ball b_{k-1} circumscribed to the vertices of the rectangular section $\mathcal{F}$ of $(\mathcal{R}o)$. Let $(\mathcal{C}o)$ be the cone with section b_{k-1} and apex s. Then,*

$$\mathcal{R}o \subset \mathcal{B}_2 \Leftrightarrow \mathcal{C}o \subset \mathcal{B}_2$$

Proof: Suppose that $\mathcal{R}o \subset \mathcal{B}_2$. Ball b_{k-1} circumscribed to $\mathcal{F}$ has radius om and center o (see Figure

2). It is therefore included in the intersection of $\mathcal{B}_1$ and $\mathcal{B}_2$. Hence, cone Co is included in $\mathcal{B}_2$. The reciprocal is straigtforward, since $\mathcal{R}o \subset Co$.

Remark: If $\mathcal{R}o$ is rotated around its symmetry axis, it is still included in Co, and hence in $\mathcal{B}_2$.

Corollary 2.2 *If s is an unfinished site inside k-rectangle $\mathcal{R}_k$, and at distance t from its boundary, then we may find a k-partition of the neighbourhood of s such that at least one of the partitioning k-cells - each with volume $\frac{\pi^{\frac{k}{2}}t^k}{k2^{\frac{k(k+1)}{2}}\Gamma\left(\frac{k}{2}+1\right)}$ - has no sites in its interior.*

Proof: If s est unfinished, Lemmas 2.1, 2.2 and Corollary 2.1 imply that we may find one open cone wedge Co_s, with vertex s and no sites in its interior. However, Co_s depends on the position of the Delaunay edge originating from s.

We shall then construct a fixed k-partition of ball $\mathcal{B}_1$ and show that, if s is unfinished, at least one of the partitioning k-cells of $\mathcal{B}_1$ has no sites in its interior.

Let $(s, \vec{\imath}_1, \ldots, \vec{\imath}_k)$ be an orthonormal system of co-ordinates. Chose in Co_s a 'rectangular' cone $\mathcal{R}o_s$ such that the $(d-2)$-faces of its section are orthogonal to the axes of the co-ordinate system. Using the spherical co-ordinates $(\rho, \alpha_1, \ldots, \alpha_{k-1})$, its equation, in the above co-ordinate system, will be of the form:

$$(\mathcal{R}o_s)\begin{cases} \rho & < & r_1 & & \\ a_1 & < & \alpha_1 & < & a_1+\frac{\pi}{2} \\ \vdots & & \vdots & & \vdots \\ a_{k-1} & < & \alpha_{k-1} & < & a_{k-1}+\frac{\pi}{2} \end{cases}$$

Notice that, if the a_i are judiciously chosen, it is possible to build a partition of ball $\mathcal{B}_1$ with 'rectangular' cones $\mathcal{R}o$.

The number of elements of the paving is equal to the number of faces of an hypercube (a cube is a regular polyedron inscribable in a ball, in all dimensions [2]). The faces of this hypercube differ little from the section of the $(\mathcal{R}o)$'s. (but are indeed different, except for $k=2$)

Let us now consider, more precisely, a paving of ball $\mathcal{B}_1$ with 'rectangular' cones $\mathcal{R}o$ with symmetry axes equal to the axes of the co-ordinate system. Imagine that we divide the section of $(\mathcal{R}o)$'s along all hyperplanes orthogonal to the axes of the referential (except the one orthogonal to its symmetry axis), and containing the center of symmetry of the section of $(\mathcal{R}o)$'s. This partition yields 2^{k-1} "cells", each of which being associated to a cone wedge (ro) with vertex s, limited by $\mathcal{B}_1$.

$$(ro)\begin{cases} \rho & < & r_1 & & \\ l_1\frac{\pi}{4} & < & \alpha_1 & < & (l_1+1)\frac{\pi}{4} \\ \vdots & & \vdots & & \vdots \\ l_{k-1}\frac{\pi}{4} & < & \alpha_{k-1} & < & (l_{k-1}+1)\frac{\pi}{4} \end{cases}$$

We have found a *fixed* and finer partition of ball $\mathcal{B}_1$ by $2k \times 2^{k-1} = k2^k$ cone wedges, each with the same volume, for symmetry reasons.

Consequently, $(\mathcal{R}o_s)$ - and hence (Co_s)-, will necessarily contain at least one partition cell (ro_j) void of sites, whatever the direction of its axis (*e.g.* set $l_i = \lceil \frac{4a_i}{\pi} \rceil$).

All cone wedges (ro) have the same volume, which is thus equal to the volume of the k-ball centered in s with radius $\frac{t}{2^{\frac{k-1}{2}}}$, divided by $(k2^k)$.

$$\mathcal{V}(ro) = \frac{\pi^{\frac{k}{2}}\left(\frac{t}{2^{\frac{k-1}{2}}}\right)^k}{\Gamma\left(\frac{k}{2}+1\right)k2^k} = \frac{\pi^{\frac{k}{2}}t^k}{k2^{\frac{k(k+1)}{2}}\Gamma\left(\frac{k}{2}+1\right)}$$

(where Γ is the second-order eulerian function).

2.3 Probability for a site to be unfinished

Lemma 2.3 *Assuming the probability density f to be quasi-uniform with bounds c_1 and c_2, consider rectangle $\mathcal{R}_k \subseteq \mathcal{U}_k$ in $DT(S \cap \mathcal{R}_k) <_\Delta DT(S)$. If $s \in S \cap \mathcal{R}_k$ is a site at distance t from the boundary of $\mathcal{R}_k$, then the probability $p(\mathcal{C}_1)$ for s to be unfinished in $DT(S \cap \mathcal{R}_k)$ wrt $DT(S)$ is, at most:*

$$p(\mathcal{C}_1) \leq k2^k\left(1 - \frac{c_1\pi^{\frac{k}{2}}t^k}{k2^{\frac{k(k+1)}{2}}\Gamma\left(\frac{k}{2}+1\right)}\right)^{n-1}$$

Proof: $(\vec{\imath}_1, \ldots, \vec{\imath}_k)$ being the same orthonormal basis as before, $\mathcal{B}_1$, centered in s with radius $\frac{t}{2^{\frac{k-1}{2}}}$, is partitioned by 'rectangular' cone wedges (ro_j), $j \in [1, k2^k]$. Using Corollary 2.2, condition $\mathcal{C}_1$ - that s is unfinished in $DT(S \cap \mathcal{U}_k)$ *wrt* $DT(S)$ - implies condition $\mathcal{C}_2$ - that at least one of the open partitioning k-cells (ro_j) is void of sites. Hence,

$$\begin{aligned} P(\mathcal{C}_1) \leq P(\mathcal{C}_2) &\leq \sum_{j=1}^{k2^k} P(S \cap (co_j)) = \emptyset) \\ &= \sum_{j=1}^{k2^k}\left(1 - \int_{(co_j)} f\right)^{n-1} \\ &\leq k2^k\left(1 - \frac{c_1\pi^{\frac{k}{2}}t^k}{k2^{\frac{k(k+1)}{2}}\Gamma\left(\frac{k}{2}+1\right)}\right)^{n-1} \end{aligned}$$

using property $f(x,y) \geq c_1$.

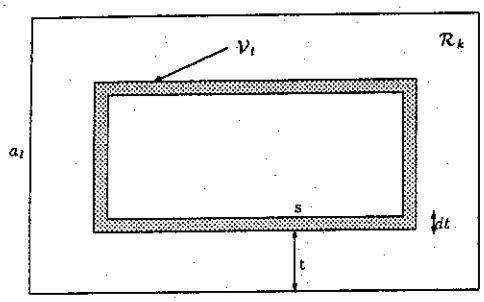


Figure 3: *Theorem 2.1*

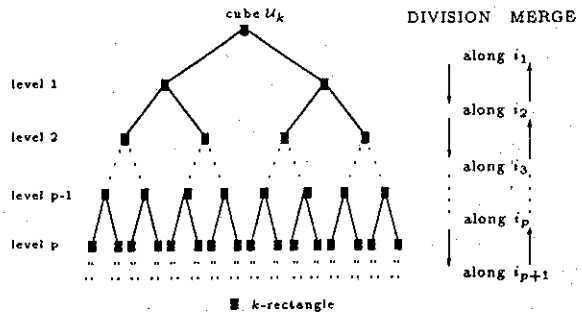


Figure 4: *Tree representation of divides and conquers according to a balanced kd-tree.*

2.4 Expected number of unfinished sites

Lemma 2.4 *For integers $n \geq 0$ and $k \geq 2$:*

$$\mathcal{I}_n^k = \int_0^1 (1-x^k)^n \, dx = \prod_{i=1}^{n} \frac{ki}{ki+1} < \frac{1}{\sqrt[k]{n+1}}$$

(proof by induction).

Theorem 2.1 *In the setting of Lemma 2.3, if $\mathcal{S}_k$ is the surface of $\mathcal{R}_k$, and $E(\mathcal{R}_k)$ is the expected number of unfinished sites in $\mathcal{R}_k$, then:*

$$E(\mathcal{R}_k) \leq \frac{n}{\sqrt[k]{n}} \frac{c_2}{\sqrt[k]{c_1}} \mathcal{S}_k \frac{k 2^{\frac{k(k+1)}{2}} \sqrt[k]{k\Gamma\left(\frac{k}{2}+1\right)}}{\sqrt{\pi}}$$

Proof: Let $a_1, a_2, \ldots, a_k$ be the lengths of the sides of k-rectangle $\mathcal{R}_k$. Let a_l be the length of the smallest side of $\mathcal{R}_k$. Consider the elementary volume $\mathcal{V}_t$ comprising the points of $\mathcal{R}_k$ at a distance between t and $t+dt$ from the boundary of $\mathcal{R}_k$ (with $0 \leq t \leq \frac{a_l}{2}$) (Figure 3). Let us call $\mathcal{S}_t$ the exterior surface of $\mathcal{V}_t$:

$$\mathcal{S}_t = 2\sum_{j=1}^{k} \left(\prod_{i \neq j} (a_i - 2t) \right) \leq \mathcal{S}_k = 2\sum_{j=1}^{k} \left(\prod_{i \neq j} a_i \right)$$

$$\Rightarrow \mathcal{V}_t \leq \mathcal{S}_t \times dt \leq \mathcal{S}_k \times dt$$

The probability for a given site to belong to $\mathcal{V}_t$ is:

$$\int_{\mathcal{V}_t} f \leq c_2 \times \mathcal{V}_t \leq c_2 \times \mathcal{S}_k \times dt$$

Using Lemma 2.3, we have the following estimation:

$$\begin{aligned} E(\mathcal{R}_k) &\leq n \int_0^{\frac{a_l}{2}} k 2^k \left(1 - \frac{c_1 \pi^{\frac{k}{2}} t^k}{k 2^{\frac{k(k+1)}{2}} \Gamma\left(\frac{k}{2}+1\right)}\right)^{n-1} c_2 \mathcal{S}_k \, dt \\ &= n \frac{c_2}{\sqrt[k]{c_1}} \mathcal{S}_k \frac{k 2^{\frac{k(k+1)}{2}} \sqrt[k]{k\Gamma\left(\frac{k}{2}+1\right)}}{\sqrt{\pi}} \\ &\quad \times \int_0^{\frac{a_l \sqrt{\pi}}{2^{\frac{k(k+1)}{2}} \sqrt[k]{k\Gamma\left(\frac{k}{2}+1\right)}}} (1-x^k)^{n-1} \, dx \end{aligned}$$

setting $x = t \sqrt[k]{\frac{c_1 \pi^{k/2}}{k 2^{\frac{k(k+1)}{2}} \Gamma\left(\frac{k}{2}+1\right)}}$.

Since $a_l \leq 1$ et $c_1 \leq 1$, we may write:

$$\begin{aligned} E(\mathcal{R}_k) &\leq n \frac{c_2}{\sqrt[k]{c_1}} \mathcal{S}_k \frac{k 2^{\frac{k(k+1)}{2}} \sqrt[k]{k\Gamma\left(\frac{k}{2}+1\right)}}{\sqrt{\pi}} \\ &\quad \times \int_0^1 (1-x^k)^{n-1} \, dx \\ &\leq \frac{n}{\sqrt[k]{n}} \frac{c_2}{\sqrt[k]{c_1}} \mathcal{S}_k \frac{k 2^{\frac{k(k+1)}{2}} \sqrt[k]{k\Gamma\left(\frac{k}{2}+1\right)}}{\sqrt{\pi}} \end{aligned}$$

using the upper bound on the integral from Lemma 2.4. Hence, $E(\mathcal{R}_k) = O(\frac{n}{\sqrt[k]{n}} \times \frac{c_2}{\sqrt[k]{c_1}} \times \mathcal{S}_k)$.

Remark: This bound is better than tat obtained by Katajainen and Koppinen in the plane [7].

3 Average-case analysis of a class of k-dimensional merge procedures using the notion of unfinished sites

In this section, we demonstrate how the previous result may be exploited to analyze certain classes of multi-dimensional *divide-and-conquer*-type algorithms. Please refer to figures 4 and 5.

Theorem 3.1 *Let S be a set of n sites distributed in a k-dimensional unit hypercube $\mathcal{U}_k$, according to a quasi-uniform density probability f, with bounds c_1 and c_2 ($c_1 \leq c_2$). Consider the following algorithmic scheme:*

Divide step: *Divide $\mathcal{U}_k$ until reaching cells, each containing one single site, using a balanced kd-tree ([1]);*

Merge step: *Then re-construct $\mathcal{U}_k$ through successive merges in reverse order (of the divisions).*

If merging two subsets takes times proportional to the number of unfinished sites (wrt $\mathcal{U}_k$), then the whole

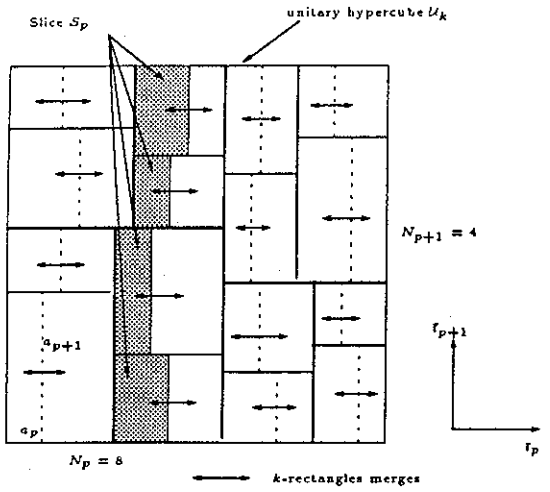


Figure 5: *Merges at level p: Slice S_p with all its (shaded) hyperrectangle constituents.*

merge *phase of the algorithm will be proportional to* n.

Proof: Using Theorem 2.1, the running time of the *merge* phase will be proportional to

$$\frac{n}{\sqrt[k]{n}} \frac{c_2}{\sqrt[k]{c_1}} \frac{k 2^{\frac{k(k+1)}{2}} \sqrt[k]{k\Gamma\left(\frac{k}{2}+1\right)}}{\sqrt{\pi}} \sum_{k-rect.} \mathcal{S}$$

Hence, we must evaluate $\sigma = \sum_{k-rect.} \mathcal{S}$, this quantity representing the sum of the surfaces of all k-rectangles involved in the merges. We shall decompose σ into sub-sums σ_p, one for each merge level p.

Let $(\vec{i}_1, \ldots, \vec{i}_k)$ be an orthonormal basis, such that each axis is orthogonal to one hyperface of $\mathcal{U}_k$. The kd-tree produces a partition of $\mathcal{U}_k$, with n k-rectangles.

Definition 3.1 *Let us define* slice $\mathcal{S}_j$ *relative to direction* $\vec{i}_j$ *as one subset of the hyperrectangles partitioning* $\mathcal{U}_k$, *such that the orthogonal projection of* $\mathcal{S}_j$ *on either hyperface of* $\mathcal{U}_k$ *orthogonal to* $\vec{i}_j$ *constitutes a partition of this hyperface.*

During the construction of the kd-tree, the division of space – orthogonally to one direction – doubles the number of slices relative to this direction. Moreover, note that the whole rectangular partition for $\mathcal{U}_k$ may be itself partitioned into slices with the same direction, whatever this direction. Let us call N_j the number of slices relative to direction $\vec{i}_j$.

Consider merge-level p. Save for the last level – for which this value is only an upper bound for N_j – a simple induction argument yields:

$$N_j = 2^{\lceil \frac{p-j+1}{k} \rceil}.$$

Let us now evaluate σ_p by grouping k-rectangles into slices, and this, relatively to all directions. Since the surface of any hyperface of $\mathcal{U}_k$ is 1, the overall sum is equal to $2\sum_{j=1}^{k} N_j$, whence:

$$\frac{1}{2}\sigma_p \leq \sum_{j=1}^{k} 2^{\lceil \frac{p-j+1}{k} \rceil} \leq k 2^{\lceil \frac{p}{k} \rceil}$$

Summing over all h merge-levels yields:

$$\frac{1}{2}\sigma = \frac{1}{2}\sum_{p=1}^{h} \sigma_p \leq \sum_{p=1}^{h} k 2^{\lceil \frac{p}{k} \rceil} = k \sum_{p=1}^{h} 2^{\lceil \frac{p}{k} \rceil}$$

$$\leq k(k \sum_{w=1}^{\lceil \frac{h}{k} \rceil} 2^w) \leq k^2 2^{\lceil \frac{h}{k} \rceil + 1} \leq 4k^2 2^{\lceil \frac{h}{k} \rceil - 1}$$

Since the division scheme is supported by a balanced kd-tree, its own height h is such that $2^{h-1} < n \leq 2^h$, and hence:

$$\sqrt[k]{n} > 2^{\frac{h-1}{k}} \geq 2^{\lceil \frac{h}{k} \rceil - 1} \Rightarrow \sigma = \sum_{k-rect.} \mathcal{S} \leq 8k^2 \sqrt[k]{n}$$

This implies that the average running time of the merge step is linear in the number of sites. It is, more precisely, proportional to:

$$n \frac{c_2}{\sqrt[k]{c_1}} \frac{k^3 2^{\frac{k(k+1)+6}{2}} \sqrt[k]{k\Gamma\left(\frac{k}{2}+1\right)}}{\sqrt{\pi}}$$

4 Application:2D Delaunay triangulation in linear expected time after two-directional sorting

The previous result may be implemented in the construction of the Delaunay triangulation of a set of planar points, after a preprocessing two-directional sort. In general, the optimized algorithm proves to be 2 to 3 times faster than the standard D&C one [8]. For n distincts sites, the total number of edges created and of edges destroyed is close to $4n$ and n, respectively, and the number of edges of the initial triangulation still valid in the final one is close to $3n$.

Extensive tests were conducted on sets of up to seven millions sites, uniformly distributed in a square domain. Both the standard algorithm, and the standard-based optimized algorithm were tested on a Silicon Graphics Indy (200 MHz), and on a Convex C3 super-computer.

We have shown, in Figure 6, the results obtained on the C3. These results strongly corroborate the theory, and show the asymptotic $O(n \log n)$ and $O(n)$ behaviour of the standard and optimized versions, respectively (excluding preprocessing sorts). Ohya, Iri and Murota ([10]) have shown that *even* the average-case running time for the standard merge step is $\Omega(n \log n)$.

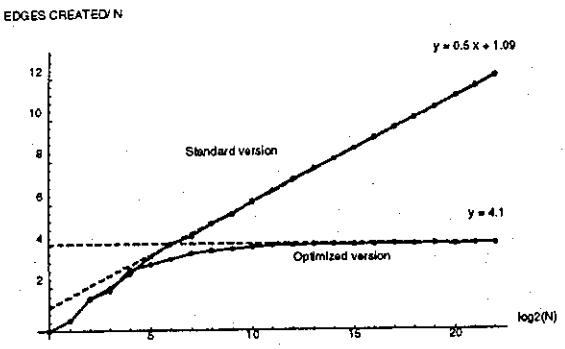


Figure 6: *Comparison of the standard (Lee and Schachter's) version, and of the optimized version.*

We have chosen to measure performance through the total number of edges created by the programs, since the running time for the triangulation is proportional to this quantity. The difference between the running times of both versions becomes perceptible from 60 sites on, and, for 130,000 sites, the optimized version is already twice as fast as the standard one. The difference increases with n.

On a 200 Mhz Indy workstation, 200,000 sites are triangulated in 7 seconds, after a sorting phase of 3 seconds. The rate of triangulation (exclusive of sorting) is about 30,000 sites per second (between 50,000 and 60,000 triangles).

5 Conclusion

The evaluation of the expected number of unfinished sites in a k-rectangle, under quasi-uniform distribution, is an interesting result in itself, that may be used to analyze various algorithms. It may be added to the probabilistic results already obtained by Bern, Eppstein and Yao ([3]) on the Delaunay triangulation in any dimension.

However, the major interest of this result lies in its application to the construction of the Delaunay triangulation in any dimension with a divide-and-conquer algorithm based upon a balanced kd-tree partition scheme. This paper has shown the efficiency of such a scheme in the planar case. The corresponding algorithm is fairly easy to implement, and seems to be the fastest, at least for quasi-uniform distributions, to our knowledge.

One of our next research goals is to prove Theorem 3.1 for other classical distributions. Doing this would give our method a neat advantage over all those based on regular grids.

Another goal is to prove that the method may be applied in k-dimensional setting, to obtain, after multi-dimensional pre-sorting, a triangulation the expected running time of which would be proportional to the number of sites. It would be interesting to compare this method with Dwyer's ([5]), in which it is suggested to use a pre-partition of the sites based on a regular grid. In $3D$, the algorithm is almost established.

Acknowledgments

The authors wish to thank Jacques Hervé for his skill, help and encouragements; Philippe Nouaille for his participation in the coding of the optimized version; Tam Vo-Dinh for making the Convex C3 computer available, and finally Jean-Noël Theillout for letting the first author go on with this work.

References

[1] J.L.Bentley. Multidimensional binary search trees used for associated searching. In *Communications of the ACM*, vol. 18, pp 509-517, 1975.

[2] M. Berger. Géométrie 3. convexes et polytopes, polyèdres réguliers, aires et volumes. Ed. *Fernand Nathan, Paris*, 176 pp, 1974.

[3] M. Bern, D. Eppstein, F. Yao. The expected extremes in a Delaunay triangulation. In *International Journal of Computational Geometry and Applications*, 1(1), pp 79–91, 1991.

[4] R.A. Dwyer. A faster divide-and-conquer algorithm for constructing Delaunay triangulations. In *Algorithmica*, 2, pp 137–151, 1987.

[5] R.A. Dwyer. Higher-Dimensional Voronoi Diagrams in Linear Expected Time. In *Discrete Comput Geom*, 6, pp 343–367, 1991.

[6] L.J.Guibas, J.Stolfi. Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams. *ACM Transactions on Graphics*, 4, pp 74–123, 1985.

[7] J. Katajainen, M. Koppinen. Constructing Delaunay triangulations by merging buckets in quadtree order. In *Annales Societatis mathematicae Polonae*, Series IV, Fundamenta Informaticae 11, pp 275–288, 1988.

[8] D.T. Lee, B.J. Schachter. Two algorithms for constructing a Delaunay triangulation. In *International Journal of Computer and Information Sciences*, 9, pp 219–242, 1980.

[9] A. Maus. Delaunay triangulation and the convex hull of n points in expected linear time. In *BIT* 24, pp 151–163, 1984.

[10] T. Ohya, M. Iti, K. Murota. Improvements of the incremental method for the Voronoi diagram with computational comparison of various algorithms. In *Journal of the Operations Research Society of Japan* 27, pp 306–337, 1984.

On-line Searching in Geometric Trees*

(Extended Abstract)

Sven Schuierer†

Abstract

In this paper we study the problem of a robot searching for a target in an unknown geometric tree with m leaves. The target can only be detected if the robot reaches the location of the target. The search cost is proportional to the distance traveled by the robot. We are interested in the *competitive ratio*, that is the ratio of the distance traveled by the robot to the length of the shortest path to reach the goal. We provide optimal upper and lower bounds for the competitive ratio of search strategies in geometric trees.

As an application of our strategy we present an algorithm to search in simple polygons. It works for arbitrarily oriented polygons and achieves a competitive ratio of $1 + en$ where n is the number of vertices of the polygon. We also present a different strategy that achieves a competitive ratio of $2n - 7$.

1 Introduction

Searching for a goal is an important and well-studied problem in robotics. In many realistic situations the robot does not know its environment in advance, i.e., it does not have a map of its surroundings [DHS95, DI94, Kle92, Kle94, LOS95, PY89]. In this paper we examine the problem of searching for a goal in unknown geometric trees as well as simple polygons.

The search of the robot can be viewed as an *on-line* problem since the robot has to make decisions about the search based only on the part of its environment that it has explored before. One way to judge the performance of an on-line search strategy is to compare the distance traveled by the robot to the length of the shortest path from its starting point s to the goal g. The ratio of the distance traveled by the robot to the optimal distance from s to g maximized over all possible starting positions s and locations of the goal is called the *competitive ratio* of the search strategy [ST85].

*This research is supported by the DFG-Project "Diskrete Probleme", No. Ot 64/8-2.

†Institut für Informatik, Universität Freiburg, Am Flughafen 17, Geb. 051, D-79110 Freiburg, FRG, e-mail: schuiere@informatik.uni-freiburg.de

One problem in this setting that is very well investigated is searching on m concurrent rays. Here a point robot is imagined to stand at the origin of m rays and one of the rays contains the goal g whose distance to the origin is unknown. The robot can only detect g if it stands on top of it. If the rays are allowed to contain branching vertices, then we obtain a geometric tree. Klein presents an algorithm to search in a geometric tree with m leaves that has a competitive ratio of $8m-3$ [Kle97]. This algorithm can be applied to search in a simple polygon and achieves a competitive ratio of $8n - 3$ if the polygon has n vertices [Ick94, Kle97].

In this paper we show how to adapt the optimal deterministic strategy to search on m concurrent rays to geometric trees, i.e., trees embedded in d-dimensional Euclidean space. We obtain an algorithm with a competitive ratio of $1 + 2m^m/(m-1)^{m-1} \leq 1 + 2em$ if the tree contains m leaves. The algorithm we obtain can be directly applied to searching in simple polygons which leads to a competitive ratio of $1+n^n/(n-1)^{n-1} \leq 1+en$ if the polygon has n vertices. We also present a different strategy with a competitive ratio of $2n - 7$.

The paper is organized as follows. In the next section we introduce some definitions. In Section 3 we present the algorithm to search in a geometric tree and analyse its competitive ratio. Section 4 shows how the algorithm can be applied to searching in a simple polygon. It also introduces a second algorithm for searching in a simple polygon. We conclude with some final remarks in Section 5.

Some of the proof are omitted due to the limited space.

2 Definitions

Let C be a simple closed curve consisting of n line segments such that no two consecutive segments are collinear. We define a *simple polygon* P to be the union of C and its interior. A vertex of P is the end point of a line segment on the boundary of P. A vertex is called convex is the subtending angle in the interior of P is less than $\pi/2$ and reflex otherwise.

Definition 2.1 *A geometric tree $T = (V, E)$ is a tree embedded into $\mathbb{E}^d$ such that each $v \in V$ is a point and each edge $e \in E$ is a polygonal path whose end points lie in V. The paths of E intersect only at points in V, and they do not induce any cycles.*

If p and q are two points inside P, then we denote the shortest path from p to q by $shp(p, q)$. The union of all shortest paths from a fixed point p to the vertices of P forms a geometric tree called the *shortest path tree* of p. It is denoted by T_p.

Let A be an algorithm to search for a target t in an environment class $\mathcal{E}$. We denote the length of the path traveled by the robot to find t using algorithm A starting at s in $E \in \mathcal{E}$ by $d_E^A(s,t)$ and the length the shortest path from the robot location s to t by $d_E(s,t)$. The competitive ratio of A is defined as

$$\max_{E \in \mathcal{E}, s,t \in E} \frac{d_E^A(s,t)}{d_E(s,t)}.$$

3 Searching in Trees

In this section we consider the problem of searching in geometric trees. Before we describe our strategy to search in trees, we briefly discuss the optimal algorithm to search for a point in m concurrent rays.

3.1 Multiway Ray Search

The model we consider is the following. The robot is placed at the meeting point or origin of m concurrent rays and it has to find a point t which is situated in one of the rays. The distance of the point t is unknown to the robot though a lower bound ε on the distance to t is known to the robot. The robot can only detect the point t when it reaches t. A deterministic strategy which achieves the optimal competitive ratio works as follows. The robot visits the rays one by one in a round robin fashion until the point t is reached. In every ray, the robot goes a certain distance and turns back if the point t is not found and explores the next ray. The number of steps from the origin the robot walks before the i-th turn is determined by the function

$$f(i) = \left(\frac{m}{m-1}\right)^i \varepsilon.$$

It is easy to see that the worst case ratio of the distance traversed by the robot to the actual distance of t from the origin (i.e., the competitive ratio of this strategy) is

$$1 + 2\frac{m^m}{(m-1)^{m-1}} \leq 1 + 2em.$$

For instance, if $m = 2$, then the competitive ratio is 9.

3.2 Geometric Trees

At first it seems that the problem of searching in geometric trees can be solved trivially by using the optimal strategy to search on m rays if the tree has m leaves. However, there is one crucial difference. Whereas in the ray searching problem the number of the rays to be searched is known in advance, the number of branch vertices of the tree and their degree is not known in advance in tree searching. Klein provides a simple way to avoid this problem by dividing the search into *phases* [Kle97]. In phase i all the known branches are explored to a depth of 2^i. The phase ends when the last branch is visited and then the search depth is doubled. It can be easily seen that the competitive ratio of this strategy is at most

$$\frac{2\sum_{i=0}^{k} m2^i + 2(m-1)2^{k+1} + 2^k}{2^k} \leq 8m - 3.$$

In the following we show that by adapting the strategies to search in m rays more carefully one can achieve a competitive ratio of at most $1+2em$. We assume that we are given a lower bound ε on the distance to the target as well as the distance to the closest branch vertex. In the applications that we present such a bound can be easily derived. Assume that in the beginning m_1 rays are incident to the root of the tree. We start off with the optimal search strategy to search m_1 rays. The initial step length is set to $d_1 = \varepsilon(m_1 - 1)^{m_1}/m_1^{m_1}$. This guarantees that every ray has been visited at least once before a branch vertex is detected.

The strategy works as follows. Assume the explored tree has m leaves $l_1, \ldots, l_m$. The robot searches the root to leaf paths by simulating the strategy to search on m concurrent rays. Let $\mathcal{P}_i$ be the path from the root to leaf l_i. The robot travels on each path $\mathcal{P}_i$ for a distance of d_i where d_i is the current step length and then returns to the origin. What happens if new edges branching off $\mathcal{P}_i$ are discovered is discussed below. The requirement that the robot should return to the origin is, of course, overly restrictive since the branching pattern of the tree often makes it unnecessary to return to the root. While in practice a robot using our algorithm of course would make use of shortcuts, it is conceptually more convenient to assume that the robot returns to the origin, i.e., to treat the paths $\mathcal{P}_i$ as separate rays.

As indicated above it may happen that the robot discovers a new edges while traveling of $\mathcal{P}_i$. Once it discovers a new edge on $\mathcal{P}_i$, it reduces its step length d_i by a factor of $(1 - 1/m^2)^m$ and increases m by one. This only happens the first time a new edge branching off $\mathcal{P}_i$ is discovered. All other new edges are just collected without affecting d_i. After it has reached the end point of the current root to leaf path, it returns to the

first point at which a new edge is detected. It chooses a newly discovered edge, increases the step length by $m/(m-1)$ and follows it. If there are other new edges remaining to be explored, the step length is again reduced by a factor of $(1-1/m^2)^m$ and m is increased. Once all unexplored branches are exhausted, the robot returns to the origin and chooses the next root to leaf path. In a more algorithmic notation the algorithm can be described as follows. The built-functions that are used in the algorithm are explained below.

Algorithm Tree-Search
Input: a geometric tree T with root r, a target t, and a lower bound ε on the distance from r to t and to the closest vertex of T
Output: a path in T from r to t

1 let $e_1,\ldots,e_m$ be the edges incident to r
2 **for** $i := 1$ **to** m **do**
3 follow e_i to the distance $d_i = \varepsilon(\frac{m-1}{m})^{m-i}$
4 let l_i be the end point of the explored part of e_i
5 $i := m+1$; $d_i := m/(m-1)$; $d_i^* := \varepsilon(\frac{m-1}{m})^{m-1}$
6 *new-edges* := *new-edges*$_i$:= [ ]
7 $L_i := [(l_1,e_1),\ldots,(l_m,e_m)]$; $m_i = m$
8 **loop**
9 **if** *new-edges* = [ ]
10 **then** $(l,\mathcal{P}) :=$ *remove-first*(L_i)
11 **else** $(l,\mathcal{P}) :=$ *remove-first*(*new-edges*)
12 **if** *new-edges* $\neq$ [ ] **then**
13 *shrink*$(L_i, 1-1/m_i^2)$
14 $d_i := (1-1/m_i^2)^{m_i} d_i$
15 follow $\mathcal{P}$ until
16 **Case 1:** t is detected ▷
17 **exit loop**
18 **Case 2:** a new vertex v is reached ▷
19 **if** *new-edges* = [ ] **then**
20 *shrink*$(L_i, 1-1/m_i^2)$
21 $d_i := (1-1/m_i^2)^{m_i} d_i$
22 $v_i^* := v$
23 let $e_1,\ldots,e_k$ be the edges incident to v
24 **for each** edge e_j **do**
25 let $\mathcal{Q}_j$ be $\mathcal{P}$ concatenated with e_j
26 $\mathcal{P} := \mathcal{Q}_1$
27 *append*(*new-edges*, $[(v,\mathcal{Q}_2),\ldots,(v,\mathcal{Q}_k)]$)
28 **go to** 15
29 **Case 3:** the distance traveled on $\mathcal{P}$ is at least d_i ▷
30 let l be the end point of $\mathcal{P}$
31 $L_{i+1} :=$ *concat*$(L_i, [(l,\mathcal{P})])$
32 **if** *new-edges* = [ ]
33 **then** return to the root r
34 $m_{i+1} := m_i$
35 **else** return to v_i^*
36 $m_{i+1} := m_i + 1$
37 *new-edges*$_{i+1}$:= *new-edges*; $v_{i+1}^* := v_i^*$
38 $d_{i+1}^* := d(l_1, r)$
where $(l_1,\mathcal{P}_1)$ is the first tuple of L_{i+1}
39 $d_{i+1} := m_{i+1}/(m_{i+1}-1)d_i$
40 $i := i+1$

Algorithm *Tree-Search* makes use of a couple of procedure calls which we explain in the following. The procedure *remove-first*(L) returns the first element of L and removes it from L. The procedure *append*(L_1,L_2) appends the list L_2 to the list L_1. The function *concat*(L_1,L_2) returns the list that is the concatenation of L_1 and L_2. (The difference between *append* and *concat* is purely syntactical.) Finally, the procedure *shrink* is described below.

Note that the index i can be removed without affecting the computation of the algorithm. It is introduced in order to simplify the analysis of the algorithm. In particular, the assignments in Steps 34 and 37 are unnecessary if the index i is removed.

Algorithm Shrink
Input: a list $L = [(p_1,\mathcal{P}_1)\ldots,(p_k,\mathcal{P}_k)]$ consisting of tuples of points p_j and paths $\mathcal{P}_j$ that start at r with p_j on $\mathcal{P}_j$ and a number α
Output: the list L where the distance of the points p_j on $\mathcal{P}_j$ to r is reduced by a factor of α^{j-1}

for $j := 1$ **to** $|L|$ **do**
let $(p_j,\mathcal{P}_j)$ be the jth element of L
move p_j on $\mathcal{P}_j$ closer to r by a factor of α^{j-1}

It is easy to see that algorithm *Tree Search* is correct since clearly all the edges of T are visited. If no new edges are discovered anymore, then the step length of the root to leaf paths increases exponentially by a factor of $m/(m-1)$ after each step. This guarantees that the target is eventually found.

3.3 Analysis of Algorithm *Tree Search*

In order to analyse the strategy we show that a number of invariants are maintained.

Invariant 1 (Algorithm *Tree Search*) *At Step 8 L_i contains $m_i - \delta$ tuples $(p_j,\mathcal{P}_j)$ with*

$$d(r,p_j) = \left(\frac{m_i}{m_i-1}\right)^{j-1+\delta} d_i^*, \qquad (1)$$

for $1 \le j \le m_i - \delta$, and

$$d_i = (\frac{m_i}{m_i-1})^{m_i-\delta} d_i^*$$

where $\delta = 0$ if new-edges$_i$ = [ ] and $\delta = 1$ otherwise.

Proof: In order to show Invariant 1 we first note that the invariant clearly holds the first time the loop is entered. In this case $d(r,p_j) = \varepsilon(\frac{m_i-1}{m_i})^{m_i-j}$, for

$1 \leq j \leq m_i$, and $d_i^* = \epsilon(\frac{m_i-1}{m_i})^{m_i-1}$. So assume that invariant is true at Step 8. First assume that *new-edges*$_i$ = []. In this case the first tuple $(l_1, \mathcal{P}_1)$ of L is removed in Step 10. After Step 31 L_{i+1} consists of the tuples $[(p_2, \mathcal{P}_2), \ldots, (p_{m_i}, \mathcal{P}_{m_i}), (l, \mathcal{P})]$. We distinguish two cases. If no new vertex v of T is encountered in Step 18 while the robot follows $\mathcal{P}$, then the distance of l to r is in Step 31

$$\begin{aligned} d(l,r) &= d_i = \left(\frac{m_i}{m_i-1}\right)^{m_i} d_i^* \\ &= \left(\frac{m_i}{m_i-1}\right)^{m_i-1} d(p_2,r). \end{aligned}$$

Since $d_{i+1}^* = m_i/(m_i-1)d_i^*$, it is easy to see that L_{i+1} again satisfies the invariant if the elements of L_{i+1} are renumbered appropriately. After Step 39 the value of d_{i+1} is

$$\begin{aligned} d_{i+1} &= \frac{m_i}{m_i-1}\left(\frac{m_i}{m_i-1}\right)^{m_i} d_i^* \\ &= \left(\frac{m_i}{m_i-1}\right)^{m_i} d_{i+1}^* \end{aligned}$$

as claimed.

If a new vertex v is encountered, then $d(p_j, r)$ is reduced by a factor of $(1-1/m_i^2)^{j-1}$, for $2 \leq j \leq m_i$, and d_i is reduced by a factor of $(1-1/m_i^2)^{m_i}$ in Step 20. By the invariant we have after Step 20

$$\begin{aligned} &\left(1-\frac{1}{m_i^2}\right)^{j-1} d(p_j,r) \\ &= \left(\frac{(m_i+1)(m_i-1)}{m_i^2}\right)^{j-1}\left(\frac{m_i}{m_i-1}\right)^{j-1} d_i^* \\ &= \left(\frac{m_i+1}{m_i}\right)^{j-1} d_i^*, \end{aligned}$$

for $2 \leq j \leq m_i$, and, similarly, $d_i = (\frac{m_i+1}{m_i})^{m_i} d_i^*$ after Step 21. After the tuple $(l, \mathcal{P})$ is added to L_{i+1} in Step 31, the invariant for L_{i+1} holds again since now *new-edges*$_{i+1} \neq$ [] and L_{i+1} contains $m_i = m_{i+1} - 1$ tuples which satisfy Condition 1. After d_i is multiplied with $m_{i+1}/(m_{i+1}-1)$ d_{i+1} also satisfies Invariant 1 again.

Note that the situation may occur that d_i is reduced so much in Step 21 that the robot has actually already traveled farther than the new value of d_i. This does not pose a problem as the robot then just remains at v and follows the next edge. If the new step length d_{i+1} is again less than $d(r,v)$, there is nothing to do and the robot considers the next edge and so on. Finally, it will encounter an edge for which d_i is larger than $d(r,v)$. For, if there have been k new edges in the beginning, then the new step length to explore the last edge is $(\frac{m_i+k}{m_i+k-1})^{m_i+k}$ which is larger than $(\frac{m_i}{m_i-1})^{m_i}$. Since $(\frac{m_i}{m_i-1})^{m_i}$ is the step length of the iteration in which the robot encountered v, $d_i \geq d(v,r)$ in the last step.

Finally, assume that *new-edges*$_i \neq$ []. In this case the first tuple $(l, \mathcal{P})$ of *new-edges*$_i$ is removed in Step 11. Moreover, since *new-edges*$_i \neq \emptyset$, L_i contains only $m_i - 1$ tuples and $d_i = (\frac{m_i}{m_i-1})^{m_i-1} d_i^*$. First assume that after $(l, \mathcal{P})$ is removed from *new-edges*$_i$, *new-edges*$_i$ = [] and no new vertex is encountered. Hence, after $(l, \mathcal{P})$ is added to L_{i+1} and d_i is increased by a factor of $m_{i+1}/(m_{i+1}-1)$ the invariant holds again.

If after the removal of $(l, \mathcal{P})$ from *new-edges*$_i$ still *new-edges*$_i \neq$ [] or *new-edges*$_i$ = [] and a new vertex is encountered by traveling on $\mathcal{P}$, then we see as above that, after L_i is shrunk and d_i is reduced, we obtain

$$\begin{aligned} d(p_j,r) &= \left(\frac{m_i+1}{m_i}\right)^{j-1} d_i^* \quad \text{and} \\ d_i &= \left(\frac{m_i+1}{m_i}\right)^{m_i} d_i^*, \end{aligned}$$

for $1 \leq j \leq m-1$. Of course, now $(p_1, \mathcal{P}_1)$ is the first tuple of L_i. And as in the second case above we see that after the tuple $(l, \mathcal{P})$ is added to L_{i+1} and m_i is incremented to $m_{i+1} = m_i + 1$ the invariant for L_{i+1} holds again; and after d_i is multiplied with $m_{i+1}/m_{i+1}-1$, d_{i+1} also satisfies Invariant 1 again. □

Note that, for each leaf l in L_i, the robot has explored the complete path to l; this is true when l is added to L_i and afterwards l is only changed in the Steps 13 and 20. In both steps l is moved closer to r. The following two invariants are stated without proof.

Invariant 2 (Algorithm *Tree Search*)
If *new-edges*$_i \neq$ [], *then at Step 8* $d(v_i^*, r) \geq \frac{m_i-1}{m_i} d_i^*$.

Invariant 3 (Algorithm *Tree Search*) *If in Step 8 the robot is at a distance of* d_{cur} *to* r, *then the total distance traveled by the robot is at most*

$$2d_i^* \sum_{j=-\infty}^{m_i-1-\delta} \left(\frac{m_i}{m_i-1}\right)^j + d_{cur}.$$

where $\delta = 0$ *if* *new-edges*$_i$ = [] *and* $\delta = 1$ *otherwise.*

3.3.1 The Competitive Ratio

Once we have established a bound on the distance traveled by the robot, it is easy to compute the competitive ratio of Algorithm *Tree Search*. So assume that the robot detects the target t. We again distinguish two cases. First assume that *new-edges*$_i$ = [] at Step 8. Then, the current search path of the robot starts at the

root r of T. By Invariant 1 the distance of l to r is d_i^*. Hence, $d(t,r) > d_i^*$. On the other hand, by Invariant 3 the distance traveled by the robot is at most

$$2d_i^* \sum_{j=-\infty}^{m_i-1} \left(\frac{m_i}{m_i-1}\right)^j + d(t,r)$$

Hence, the competitive ratio is given by

$$\begin{aligned}
&\frac{2d_i^* \sum_{j=-\infty}^{m_i-1} \left(\frac{m_i}{m_i-1}\right)^j + d(t,r)}{d(t,r)} \\
&\leq 1 + 2\frac{d_i^*(m_i-1)\left(\frac{m_i}{m_i-1}\right)^{m_i}}{d_i^*} \\
&= 1 + 2\frac{m_i^{m_i}}{(m_i-1)^{m_i-1}} \\
&\leq 1 + 2\frac{m^m}{(m-1)^{m-1}}
\end{aligned}$$

where m is the number of leaves of T.

Now assume that $new\text{-}edges_i \neq [\,]$ at Step 8. Then, the current search path of the robot starts at the vertex v_i^*. By Invariant 2 the distance of v_i^* to r is at least $\frac{m_i-1}{m_i} d_i^*$. Hence, $d(t,r) > \frac{m_i-1}{m_i} d_i^*$. On the other hand, by Invariant 3 the distance traveled by the robot is at most

$$2d_i^* \sum_{j=-\infty}^{m_i-2} \left(\frac{m_i}{m_i-1}\right)^j + d(t,r).$$

Hence, the competitive ratio is given by

$$\begin{aligned}
&\frac{2d_i^* \sum_{j=-\infty}^{m_i-2} \left(\frac{m_i}{m_i-1}\right)^j + d(t,r)}{d(t,r)} \\
&\leq 1 + 2\frac{d_i^*(m_i-1)\left(\frac{m_i}{m_i-1}\right)^{m_i-1}}{\frac{m_i-1}{m_i} d_i^*} \\
&= 1 + 2\frac{m_i^{m_i}}{(m_i-1)^{m_i-1}} \\
&\leq 1 + 2\frac{m^m}{(m-1)^{m-1}}
\end{aligned}$$

as above.

The above algorithm is optimal since m concurrent line segments also form a geometric tree. By making the line segments longer and longer, the lower bound on the competitive ratio of searching in the m concurrent line segments approaches $1 + 2m^m/(m-1)^{m-1}$ [BYCR93, Gal80].

4 Searching in Simple Polygons

In this section we present an algorithm to search for a target t inside a simple polygon P that is not known to the robot. It is based on an approach by Icking [Ick94, Kle97]. We assume that the robot is equipped with a vision system that allows it to compute its visibility polygon in P and also to recognize t. We assume that the robot is located in the beginning at the points s in P.

Let T_s be the shortest path tree of s. If t is not visible from s, then $shp(s,t)$ intersects a number of vertices. Let v be the last vertex that is intersected by $shp(s,t)$. Clearly, the line segment l from v to t is contained in P. Hence, once the robot has reached v, then it sees t and moves directly to t. Hence, if $d_A(s,v)$ denotes the length of the path that the robot travels in order to reach the vertex v using algorithm A, then the competitive ratio of A is given by

$$\frac{d_A(s,v) + d(v,t)}{d(s,v) + d(v,t)} \leq \frac{d_A(s,v)}{d(s,v)}.$$

Hence, we only need an algorithm to reach the vertices of P. This can be easily achieved by applying Algorithm *Tree Search* to T_s. That T_s is not known in the beginning is not a problem since the following lemma holds.

Lemma 4.1 ([Kle97]) *If the robot has seen the points p and q in P, then it can compute $shp(p,q)$.*

Hence, the robot can construct T_s on-the-fly as it executes Algorithm *Tree Search*. This immediately leads to a competitive ratio of $1 + 2n^n/(n-1)^{n-1} \leq 1 + 2en$ for searching for a target in simple polygons which considerably improves the $(8n-3)$-competitive algorithm of Icking [Ick94, Kle97].

However, the analysis of the above algorithm can be further improved by the following observation.

Lemma 4.2 *If s and t are two points in P, then the last vertex v of P that is intersected by $shp(s,t)$ and that is different from t is an internal node of T_s.*

Hence, in order to find the vertex v it is not necessary to consider the leaves of T_s. If we prune the m leaves of the tree, then the remaining tree T_s' can have no more than m leaves. Therefore, the number of leaves m' of T_s' is at most $n/2$. If we apply Algorithm *Tree Search* to T_s' we obtain a strategy to search in polygons with a competitive ratio of $1 + en$. Note that since we can compute shortest path between two points, it is easy to decide if a vertex is a leaf of T_s or not. If we denote the number of reflex vertices of P by r, then the competitive ratio is $1 + 2r^r/(r-1)^{r-1} \leq 1 + 2er$ since the vertices of T_s' obviously only consist of reflex vertices.

Simulating Dijkstra's algorithm on T_s' yields an even better algorithm with a competitive ratio of $1+2(r-1)$. Here, the robot repeatedly chooses the closest unvisited

vertex v, follows $shp(s,v)$, and returns if t is not visible from v.[1] Note again that even so P is not known to the robot, the closest unvisited vertex is visible to the robot. The competitive ratio is now just

$$\begin{aligned}&\frac{2\sum_{v' \text{ visited}} d(s,v') + d(s,v)}{d(s,v)}\\ &\leq\ 1 + 2|\{v' \mid v' \text{ is visited}\}|\\ &\leq\ 1 + 2(|T_{s'}| - 1)\ \leq\ 1 + 2(r-1)\end{aligned}$$

since $d(s,v) \geq d(s,v')$, for visited vertices v'.

If $|T_s'| \geq m'e/2$, then Algorithm *Tree Search* yields a better competitive ratio. A simple example are spiral polygons where Algorithm *Tree Search* achieves a competitive ratio of 9 in the worst case whereas the simulation of Dijkstra's algorithm on T_s' can have a competitive ratio of up to $2n-7$. This is also an upper bound since there are at least three convex vertices in a simple polygon. In general, the application of Algorithm *Tree Search* to T_s' has a competitive ratio of $1 + 2(2k)^{2k}/(2k-1)^{2k-1} \leq 1 + 4ek$ in polygons whose boundary can be decomposed into k convex and k reflex chains.

5 Conclusions

We present the Algorithm *Tree Search* to search for a target in an unknown geometric tree T. Algorithm *Tree Search* simulates the optimal search strategy to search in m concurrent rays and achieves an optimal competitive ratio of $1 + 2m^m/(m-1)^{m-1}$ if m is the number of leaves of T.

In the second part we show how to apply the algorithm to the problem of searching for a target in unknown simple polygons. We present two algorithms, one based on Algorithm *Tree Search* and the other based on a simulation of Dijkstra's algorithm. The first algorithm achieves a competitive ratio of $1 + en$ whereas the second achieves a competitive ratio of $2n-7$. However, if it is known in advance that the polygon can be decomposed into k convex and reflex chains, then the search algorithm based on Algorithm *Tree Search* achieves a competitive ratio of $1 + 4ek$ whereas the competitive ratio of the algorithm based on the simulation of Dijkstra's algorithm may be arbitrarily high.

The problem of searching in polygons is far from solved. There is a lower bound of $n/2$ for the competitive ratio which still leaves a significant gap compared to algorithms presented above.

[1] The same algorithm (only applied to T_s) was independently discovered by I. Semrau [Sem97].

References

[BYCR93] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106:234–252, 1993.

[DHS95] A. Datta, Ch. Hipke, and S. Schuierer. Competitive searching in polygons—beyond generalized streets. In *Proc. Sixth Annual International Symposium on Algorithms and Computation*, pages 32–41. LNCS 1004, 1995.

[DI94] A. Datta and Ch. Icking. Competitive searching in a generalized street. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 175–182, 1994.

[Gal80] S. Gal. *Search Games.* Academic Press, 1980.

[Ick94] Ch. Icking. *Motion and Visibility in Simple Polygons.* PhD thesis, Fernuniversität Hagen, 1994.

[Kle92] R. Klein. Walking an unknown street with bounded detour. *Comput. Geom. Theory Appl.*, 1:325–351, 1992.

[Kle94] J. M. Kleinberg. On-line search in a simple polygon. In *Proc. of 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 8–15, 1994.

[Kle97] R. Klein. *Algorithmische Geometrie.* Addison-Wesley, 1997.

[LOS95] A. López-Ortiz and S. Schuierer. Going home through an unknown street. In S. G. Akl, F. Dehne, and J.-R. Sack, editors, *Proc. 4th Workshop on Algorithms and Datastructures*, pages 135–146. LNCS 955, 1995.

[PY89] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. In *Proc. 16th Internat. Colloq. Automata Lang. Program.*, volume 372 of *Lecture Notes in Computer Science*, pages 610–620. Springer-Verlag, 1989.

[Sem97] I. Semrau. Personal communications, 1997.

[ST85] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

Biased search and k-point clustering

Binay K. Bhattacharya
School of Computing Science
Simon Fraser University
Burnaby, B.C., Canada, V5A 1S6

Hossam ElGindy
Department of Computer Science
The University of Newcastle
Callaghan 2308, NSW, Australia

April 28, 1997

Abstract

We have shown that some k-point clustering problems can be solved efficiently by a biased search technique. Biased search is useful if it takes more time to determine whether the optimal parameter, we are looking for, is greater than the search value than the time when the optimal parameter is less than or equal to the search value. The proposed algorithms generate the search keys randomly. All other steps of the algorithms are deterministic. We are able to achieve $O(\log k)$ improvement with high probability over the running times of the deterministic algorithms.

1 Introduction

We consider a clustering problem which can be stated in general terms as follows:

Given a set S of n points in d-dimensional space; an integer k, $1 \leq k \leq n$; and a dissimilarity measure, find a subset of S of size k (called a k-set) that minimizes the dissimilarity measure.

A number of papers have considered this form of k-point clustering problem. Dissimilarity measures that have been considered include euclidean diameter [1, 5, 2], L_∞ diameter [5, 2], circumradius [1, 4, 2, 5, 8] etc. [See [2, 5] for other references.] Suppose t^* denotes the optimal dissimilarity measure of an optimal k-set. Most algorithms for the k-point clustering problems can be modeled as follows (again in general terms):

Algorithm Cluster (S)

1. Select a value t. Set $l \leftarrow 1$.
2. For each point $s_j, j = l, 2, ..., |S|$, do the following.
   (a) Find the largest cardinality subset $W_S(s_j, t)$ of S such that $s_j \in W_S(s_j, t)$ and the dissimilarity measure of $W_S(s_j, t)$ is at most t.
   (b) If $|W_S(s_j, t)| \geq k$ then
   select a smaller t, set $l \leftarrow j$, and go to step 2 { Here $t^* \leq t$. }
   If no such t exists, $t^* = t$ and stop.
3. { Comment: There is no k-set of dissimilarity measure $\leq t$ }
   Select a larger t and go to step 2. { $t^* > t$ }.

There are two commonly used searching methods for t^*. One is the standard binary search technique and the other is the parametric search technique proposed by Megiddo [9]. The binary search technique is commonly used when the potential candidate set of t^*, say $C(S)$, is small. When $C(S)$ is a large set, Megiddo's parametric search is often used. In algorithm *Cluster* if T(n) represents the sequential time needed to execute step 2 (a), the binary search technique will determine an optimal k-set in $O(nT(n)\log n + G_S(n))$ time in the worst case where $G_S(n)$ denotes the time needed to generate the search keys for the binary search. Again if step 2 (a) can be implemented in $B(n)$ parallel steps requiring P processors, parametric search technique gives an $O(n.P.B(n) + n.B(n).T(n).\log P)$ implementation of algorithm *Cluster*. Notice

here that $G_S(n)$ term is not involved here. In most cases the second term dominates the running time. However, Megiddo's parametric search is not feasible in practice. As a matter of fact we are not aware of any non-trivial implementation of parametric search. Therefore, our objective in this paper is to avoid the parametric search of Megiddo completely in the design of our algorithms.

In algorithm *Cluster*(S), when we are selecting t, we have no control on whether $t^* \leq t$ or $t^* > t$. We call this type of search as unbiased search. It is inexpensive to determine if $t^* \leq t$ than determining if $t^* > t$. In order to determine that $t^* > t$ for a given t, we need to show in Step 2(a) that for each point q of S, $|W_S(q,t)| < k$. However, to determine $t^* < t$ for a given t, we need just an existence of one point s_j of S for which $|W_S(s_j,t)| \geq k$. In this paper we show that it is possible to take advantage of this property by biasing our search and obtain faster algorithms to solve k-point clustering problems for the dissimilarity measures such as euclidean diameter, L_∞ diameter and circumradius. The search keys during the search will be generated randomly. We achieve $O(\log k)$ running time improvement with high probability on most of the results of Eppstein and Erickson [5] and Datta et al. [2]. In some cases we also get a significant improvement on storage space requirement. Matousek [8] presented a completely different randomized algorithm which achieve the same improved space and time bound for the selection of k-point subset minimizing the circumradius. His approach can also be applied to obtain $O(\log k)$ running time improvement for the selection of k-point subset minimizing euclidean diameter and L_∞ diameter. In [8], not only the the search keys were generated randomly, the data point to be processed next, as stated in Step 2 of *Cluster*, was also selected randomly.

In section 2 we present a biased search algorithm for a generic problem. k-point clustering problems minimizing euclidean diameter and circumradius dissimilarity measures can easily be formulated from the generic problem. For all the three clustering problems we generate search keys randomly. In section 3 we describe methods to generate random search keys for the three problems being considered here. Section 4 contains the solutions of the clustering problems.

2 Generic problem

In this section we consider k-point clustering problems for the dissimilarity measures (i) euclidean diameter and (ii) circumradius. However, we explain the biased search technique for the following generic problem. Then the actual algorithm for the specified dissimilarity measure can easily be designed from the algorithm for the generic problem.

The generic problem is:

Given a set S of n points in d-dimensional space and a dissimilarity measure, determine a k-subset of S that minimizes the dissimilarity measure.

Datta et al. [2] showed that for the dissimilarity measures being considered here, the above generic problem can be reduced to $O(n/k)$ similar subproblems of size $O(k)$. The solution to the original problem is then the solution of a subproblem with the smallest dissimilarity measure. This reduction process requires $O(n \log n)$ time and $O(n)$ space.

Let S_i be the set of points of S for the i^{th} subproblem where $|S_i| = n_i$ is $O(k)$, $i = 1, 2, ..., m$; m $(\in O(|S|/k))$ being the number of subproblems. Let t^* be the dissimilarity measure of the optimal k-set to be computed. Initially, let $C(S)$ be the set of potential candidates of t^*. Let t and t' be two elements of $C(S)$. As mentioned before, let $W_{S_i}(p,t)$ be the largest cardinality subset of S_i containing the element p, with dissimilarity measure at most t. We can now show that that

Lemma 1 *For $t \leq t'$, if $|W_{S_i}(p,t)| \geq k$ and $|W_{S_j}(q,t')| < k$, for any j, q cannot be an element of the optimal k-set of S_j.*

We now formally describe our biased search algorithm. In the following we assume that t^* always lies in $(t_l, t_r]$. Initially $t_l = 0$ and $t_r = \infty$. Let N denote the number of elements of $C(S)$ lying in $(t_l, t_r]$. We assume that N is finite.

Algorithm Cluster-modified (S)

1. Reduce the k-point clustering problem on S to k-point clustering problems on S_i, $i = 1, \ldots, O(\frac{n}{k})$ where $|S_i| \in O(k)$ for each i.
   { Let s_{ij}, j= 1,2, ..., $|S_i|$ be the points of S_i. }.

2. Set starting_ subproblem $\leftarrow 1$; $t^* \leftarrow \infty$.
   Set $start[j] \leftarrow 1$ for each subproblem j.

3. Set $c_{(0)} \leftarrow t_l$.
   Select $c_{(j)}, j = 1, 2, ..., |S|^\epsilon$ elements of $C(S)$ lying in $(t_l, t_r]$, $0 < \epsilon < 1$, sorted in increasing order.
   If there are less than $|S^\epsilon|$ points of $C(S)$ in $(t_l, t_r]$, we take $c_{(1)} = c_{(2)} = \ldots = t_l$ appropriately.

4. Set $i \leftarrow$ *starting_subproblem*; $j \leftarrow start[i]$; $u \leftarrow |S|^\epsilon$.

5. Determine $W_{S_i}(s_{ij}, c_{(u)})$.

6. If $|W_{S_i}(s_{ij}, c_{(u)}| \geq k$ then set $t_r \leftarrow c_{(u)}$; $u \leftarrow u - 1$; $start[i] \leftarrow j$; *starting_subproblem* $\leftarrow i$ and go to step 5,
   else set $j \leftarrow j + 1$ and go to step 5 if $j \leq |S_i|$.

7. Set $i \leftarrow i + 1$; $j \leftarrow start[i]$ and go to step 5 if $i \leq m$.
   { m is the number of subproblems.}

8. { t^* lies in $(c_{(u-1)}, c_{(u)}]$. }
   Set $t_l \leftarrow c_{(u-1)}$. If number of elements of $C(S)$ in $(t_l, t_r]$ is one then $t^* \leftarrow t_r$ and **stop**.
   Otherwise go to step 3.

We first consider the correctness of the algorithm. We are assuming that $t^* \in (0, \infty)$ is an element of the starting potential candidate set $C(S)$. In step 6, as a consequence of Lemma 1, when $W_{S_i}(s_{ij}, c_{(u)}) \geq k$, all the points of the sets $S_1, S_2, \ldots, S_{i-1}$ and the points $s_{i1}, s_{i2}, \ldots, s_{i,j-1}$ of S_i can be eliminated. The interval containing t^* gets smaller as the algorithm progresses. This searching technique is a generalization of the standard binary search technique. Thus algorithm *Cluster-modified* correctly converges to the optimal solution t^*.

We now estimate the running time of *Cluster-modified*. Let $G(|S|)$ denote the time taken to generate all the search keys of S used in *Cluster-modified*. Step 1 requires $O(|S| \log |S|)$ time. The remaining steps except Step 5 are straightforward and clearly do not dominate the running time. Let $T(|S_i|)$ be the time required to solve step 5. The exact value of $T(|S_i|)$ is dependent on the dissimilarity measure being considered. The remaining issue to be resolved is the number of times step 5 is executed. We claim that

Lemma 2 *Step 5 is executed at most* $O(|S|^\epsilon + |S|)$ *times.*

Proof: The search tree in algorithm *Cluster – modified* is an $|S|^\epsilon$-ary tree and therefore, has a constant depth. At each level we search the keys in descending order. At every level of the search tree, therefore, there will be just one instance of the case when $c_{(u)} < t^*$ ("*Less*" instance) and at most $|S|^\epsilon$ instances of the case when $c_{(u)} \geq t^*$ ("*More*" instance). Due to Lemma 1, only a constant number of "*Less*" instances per data point per subproblem are possible during the entire search process. Hence there will be $O(|S|)$ "*Less*" instances in total. According to our search there can be at most $O(|S|^\epsilon)$ "*More*" instances and each "*More*" instance requires one call to step 5. Hence step 5 is executed at most $O(|S|^\epsilon + |S|)$ times. □

As a consequences of lemmas 1 and 2,

Lemma 3 *The algorithm* **Cluster-modified** *takes* $O(G(|S|) + |S|T(|S_i|) + |S| \log |S|)$ *time.*

We have not discussed the space complexity yet. It is very much dependent on the way the probe sequences are generated.

3 Random generation of search keys

We now discuss specific methods to generate search keys for biased searching to solve the k-point clustering problem (i.e. to determine $G(|S|)$).

Let $C(S)$ be the initial active set of potential candidates (also called events) of t^*. We are assuming here that the size of $C(S)$ is some polynomial of $|S|$ of fixed degree. Our search tree is a $|S|^\epsilon$-ary tree where

$0 < \epsilon < 1$. The tree has a constant depth. Therefore, $O(|S|^\epsilon)$ total events are generated for the entire search process. As mentioned earlier, we will generate these events randomly.

Suppose we know that t^* lies in the interval $(t_l, t_r]$. Let there be N events which lie in the interval $(t_l, t_r]$. We repeatedly generate a random event from $C(S)$ and check if they define an event in $(t_l, t_r]$. If not we repeat it. Expected number of trials needed to generate one random event in $(t_l, t_r]$ is given by the following well known lemma.

Lemma 4 *An event in $(t_l, t_r]$ can be found with high probability after $O(|C(S)| \log |C(S)|/N)$ trials.*

$O(|S|^\epsilon)$ random events in $(t_l, t_r]$ also satisfy the following property.

Lemma 5 *If $|S|^\epsilon$ event points in $(t_l, t_r]$ are randomly generated, there are no more than $O(N \log N/|S|^\epsilon)$ event points between any two consecutive event points with probability exceeding $1 - N^{-c}$ for any fixed constant $c > 0$.*

Clearly, when N is small, we will require a large number of trials to find an event. Hence we assume for now that we have an oracle which can determine in $O(U(|S|))$ time whether there exist at least M events of $C(S)$ in $(t_l, t_r]$. We also assume that the oracle returns all the events in the interval $(t_l, t_r]$ in $O(U(|S|))$ time when the number falls below M. When the number of events in the interval falls below M, the events for the searching step can be generated deterministically in $O(M \log M)$ time where all the event points in $(t_l, t_r]$ are enumerated first. Therefore,

Lemma 6 *If the events are generated randomly, with high probability the search process terminates using $O(|S|^\epsilon)$ probes and these probes can be generated in $O(U(|S|) + M \log M + |S|^\epsilon \times |C(S)| \log |C(S)|/M)$ time.*

We now discuss below the way the random events are generated for specific dissimilarity measures.

3.1 Euclidean diameter

The search space is the interpoint distances. There are $O(n/k)$ subproblems, each of size $O(k)$. Therefore, $|C(S)| \in O(nk)$. We take M to be n. We can determine the number of events in the interval $(t_l, t_r]$ in $O(nk)$ time. Hence

Lemma 7 *All the $O(n^\epsilon), 0 < \epsilon < 1$, events for the search can be found in $O(nk + n \log n)$ time. The space complexity is $O(n)$.*

3.2 Circumradius

The optimal *$k-$set* of S in d-space with the minimum circumradius must be determined by either $2, 3, ...,$ or $d+1$ data points. Hence $|C(S)| = O(nk^d)$. During the search process we will consider each case separately. Here we only consider the case when the optimal k-set is determined by $d+1$ points. The remaining cases can be solved exhaustively in $O(nk^{d-1})$.

We describe an algorithm which determines whether the number of event points in the interval $(t_l, t_r]$ is less than M. The algorithm is formally described as follows:

1. For each point p of each subproblem S_i do the following:
   (a) For each $(d-2)$ points (say V) of S_i do the following:
      i. Determine the 2-dimensional space W which is equidistant to $V \cap \{p\}$.
      ii. For each q of the remaining points of S_i determine the line segments (at most 2) which are equidistant to $V \cup \{p, q\}$ and the distance of any point on the line segment to p lies in the interval (t_l, t_r) . { There are $O(k)$ line segments in total }.
      iii. Determine all the intersection points of the line segments computed in the previous step. If the total number of intersections computed so far is $\geq M$, then stop, otherwise, report the events of $C(S)$ in the interval $(t_l, t_r]$.

We can report r intersections among $O(k)$ line segments in a plane in $O(k \log k + r \log k)$ time. Therefore $U(|S|)$ is bounded above by $O(nk^{d-1} \log k + M \log k)$. The space requirement is $O(n + M)$. From Lemma 6 it follows that

Lemma 8 *If $k \in o(n)$, n^ϵ random events in a given interval $(t_l, t_r]$ can be determined in $O(nk^{d-1} \log k)$ time and $O(n)$ space (M here is n.). If $k \in O(n)$, the space requirement increases to $O(n^{1+\epsilon})$ while the running time remains the same (M here is $n^{1+\epsilon}$).*

3.3 L_∞ diameter

We can surely generate the events the same way as we did for the euclidean diameter i.e. in $O(nk + n \log n)$ time. But in L_∞ metric we can do much better. The event space can be stored in d triangular matrices of coordinate differences with each row and column elements appear in sorted order. Here d represents the dimension of the data set S. We assume that d is fixed. We do not actually build the matrices. But we can access any entry in $O(1)$ time. During the search process we will treat each matrix separately.

We can determine the number of events in an interval $(t_l, t_r]$ in $O(n)$ time very easily [6]. These events can be represented by a set of monotone polygons requiring $O(n)$ storage space. We can then pick a random event point in $(t_l, t_r]$ in $O(\log n)$ time after $O(n \log n)$ processing by applying the standard random number generation technique (see [3]). Therefore,

Lemma 9 *All the $O(n^\epsilon)$, $0 < \epsilon < 1$, random event points for the search can be generated in $(n \log n)$ time. The space complexity is $O(n)$.*

4 Optimal k-point subset

4.1 minimizing euclidean diameter

For the euclidean diameter dissimilarity measure we can show that

Theorem 2: The k-point clustering problem that minimizes euclidean diameter can be solved in $O(nk^2 \log k)$ time and $O(n)$ space when $d = 2$ and in $O(n \log n + 2^{O(k)} n)$ time and $O(n + k^2)$ space when $d > 2$.

Proof: Eppstein and Erickson [5] have described an algorithm that takes $O(k^2 \log k)$ time and $O(k)$ space to compute $W_{S_i}(s_{ij}, c_u)$. Hence $T(|S_i|)$ is $O(k^2 \log k)$. Lemma 7 states that the required probe sequence can be generated in $O(nk + n \log n)$ time. This is the value for $G(|S|)$. The space complexity is $O(n)$. Hence, when $d = 2$, the theorem follows by substituting $G(|S|)$ and $T(|S_i|)$ in Lemma 3.

In higher dimensions, there is no efficient algorithm similar to Lemma 8. Using a brute force like approach, it is possible to determine in $O(2^{O(k)}k)$ time whether there exists a $k-set$ containing a specified point whose diameter is at most t ([5]). The storage space requirement is $O(k^2)$. Hence we can conclude that the k-point clustering problem minimizing euclidean diameter can be solved in $O(n \log n + 2^{O(k)} n)$ time requiring $O(n + k^2)$ space. □

4.2 Minimizing circumradius

Eppstein and Erickson [5] presented an $O(k^{d-1} \log k)$ algorithm to determine $W_{S_i}(s_{ij}, c_{(u)})$. The algorithm has $O(k)$ storage space complexity. Therefore from Lemma 3 and Lemma 9 we conclude that

Theorem 3: Minimum circumradius k-point subset of a set of points in d-space can be determined in $O(nk^{d-1} \log k + n \log n)$ time and $O(n)$ space when $k \in o(n)$ or in $O(nk^{d-1} \log k + n \log n)$ time and $O(n^{1+\epsilon})$ space, $0 < \epsilon < 1$, when $k \in O(n)$.

4.3 Minimizing L_∞ diameter

We can solve the k-point clustering problem minimizing L_∞ diameter in the same way as we have done for the k-point clustering problem minimizing euclidean diameter. However, we can do much better.

Overmars and Yap [10] presented an $O(k^{d/2} \log k)$ time and $O(k^{d/2})$ space algorithm (based on space sweep) for finding an optimal placement of an L_∞ box of size $c_{(u)}$ in S_i. This problem is equivalent to the

problem of finding the deepest point in an arrangement of hypercubes where each point x of S_i is replaced by a square box of size $c_{(u)}$ with x as its center. However, [10] cannot be used for the planar point set. In 2-dimensional space the algorithm of Lee [7] (also based on plane sweep) can be used to solve the problem in $O(k \log k)$ time and $O(k)$ space. The probe sequence can be generated in $O(n \log n)$ time (Lemma 9). Therefore from Lemma 3 it follows that

Theorem 4: When k is $o(n)$, k-point subset of S with the minimum L_∞ diameter can be computed in $O(n \log n + nk^{d/2-1} \log k)$ time and uses $O(n + k^{d/2})$ space for all fixed $d \geq 2$.

5 Conclusions

In this paper we showed that some k-point clustering problems can be solved efficiently by applying the biased search technique. Biased search is useful if it takes more time to determine when $t^* > t$ than when $t^* \leq t$ for any search key t. The expected running times, based on biased search, are almost always better with high probability (by $O(\log k)$ factor) than the deterministic algorithms of Eppstein and Erckson [5] and Datta et al. [2]. In some cases the storage space improvement is quite significant.

Our algorithms can also be used to improve by $O(\log k)$ factor, with high probability, the running times of the semi-dynamic/dynamic k-point clustering algorithms discussed in [2].

Matousek [8] also presented a different randomized algorithm which can be extended to achieve the same improved space and time bound as ours. Selecting search keys are the only randomization step in our algorithm. However, it is not so in [8]. We feel that there are other applications where biased search approach could be applied successfully.

References

[1] A. Aggarwal, H. Imai, N. Katoh and S. Suri. "Finding k points with minimum diameter and related problems", *J. Algorithms*, Vol. 12, 1991, 38-56.

[2] A. Datta, H.-P. Lenhof, C. Schwartz and M. Smid. "Static and dynamic algorithms for k-point clustering problems", In *Proc. 3rd Workshop Algorithms Data Struct.*, pp. 265-276. Lecture Notes in Computer Science, Vol. 709. Springer-Verlag, New York, 1993.

[3] L. Devroye. Non-uniform random variate generation. Springer-Verlag, New York, 1986.

[4] A. Efrat, M. Sharir and A. Ziv,. "Computing the smallest k-enclosing circle and related problems", In *Proc. 3rd Workshop Algorithms Data Struct.*, pp. 325-336. Lecture Notes in Computer Science, Vol. 709. Springer-Verlag, New York, 1993.

[5] D. Eppstein and J. Erickson. "Iterated nearest neighbors and finding minimal polytopes", *Discrete and Computational Geometry*, Vol. 11, 1994,321-350.

[6] G. N. Frederickson and D. B. Johnson. "The complexity of selection and ranking in $X+Y$ and matrices with sorted rows and columns", *J. Comput. System Sci.* Vol. 24, 1982, pp. 197-208.

[7] D. T. Lee. " Maximum clique problems of rectangle graph", In *Advances in Computing Research, Ed. F. P. Preparata*, pp. 91-107, Jai Press, 1983.

[8] J. Matousek. "On enclosing k points by a circle", *Information Processing Letters*, Vol. 53, 1995, 217-221.

[9] N. Megiddo. "Applying parallel computation algorithms in the design of serial algorithms", *J. Assoc. Comput. Mach.*, Vol. 30, 1983, pp.852-865.

[10] M. H. Overmars and C.-K. Yap. "New upper bounds in Klee's measure problem", *SIAM J. Comput.*, Vol. 20, 1991, 1034-1045.

Walking in the visibility complex with applications to visibility polygons and dynamic visibility

RIVIÈRE Stéphane *

1 Introduction

The visibility complex is a data structure that encodes all visibility relations between objects of a scene in the plane. However, in some applications only a subset of these informations is relevant at a time, in particular visibility informations about rays issued from (resp. going to) a given object O.

We first consider the set of faces of the complex representing rays issued from a given object. We define an order on these faces and show, once the visibility complex is computed, how to visit all of them with no additional data structure in optimal time $O(k_f)$, where k_f is the number of faces visited. Then we show how to use this walk to perform a topological sweep of the vertices incident to these faces in optimal time $O(k_v)$, where k_v is the number of these vertices, still with no additional data structure.

We next consider the set of faces of the complex representing rays issued from the "blue sky" and passing through a line segment s which is outside the convex hull of the scene. We show that the previous algorithms can be adapted simply in this new context, and we use these walks for two applications. First, we show how to compute the visibility polygon of a line segment $s = (p, q)$ outside the convex hull of the scene in $O(vis(s)+t_v(p)+t_v(q))$ time, where $vis(s)$ is the size of the visibility polygon and $t_v(p)$ (resp. $t_v(q)$) is the time to compute the view around p (resp. q). Second, we show how to maintain the view around a point moving from p to q. Once the view around p is computed, the algorithm has a total running time $O(\max(v(p), v(p, q)))$, where $v(p)$ is the size of the view around p and $v(p, q)$ the number of changes of visibility along (p, q). This algorithm is an alternative to those we have described in [Riv97b].

*iMAGIS-GRAVIR/IMAG - BP 53
38041 GRENOBLE CEDEX 09 - FRANCE
e-mail: Stephane.Riviere@imag.fr
www: http://www-imagis.imag.fr/Membres/Stephane.Riviere
iMAGIS is a joint project of CNRS/INRIA/INPG/UJF

2 The visibility complex

Visibility computations involve determining the object seen along directions of vision, that is along maximal free line segments (line segments of maximal length in free space), that we also call *rays*. Recomputing each time the object seen along a ray can be too time consuming for some visibility problems. One solution to this issue is to classify rays according to their visibility: To find the object seen along a ray, we then just have to identify the set of rays that contains the given ray and to read the visibility properties of this set.

Pocchiola and Vegter [PV96] have devised a new data structure, the *visibility complex*, which represents sets of rays having the same visibility properties. A ray going from an object O_l to another object O_r being characterized by its label (O_l, O_r), the visibility complex is the quotient space of the space of rays under the following relation $\sim$: $r_1 \sim r_2$ iff r_1 can be moved continuously to r_2 while keeping the same label.

We have adapted this structure, created initially for scenes of convex curved objects, for polygonal scenes: Each side of a polygon is a distinct object (there is an additional object O_∞ which represents the "blue sky" surrounding the scene). The visibility complex is composed of three types of elements: faces (2D components), edges (1D components representing rays passing through a polygon vertex), and vertices (0D components representing rays passing through two polygon vertices, that is edges of the visibility graph). We say that an element of the complex representing rays of label (O_l, O_r) has itself a label (O_l, O_r).

These elements are best handled by means of a duality relation, which maps a line l of the scene into a point l^* in a dual space, and maps all the lines passing through a point p in the scene into a dual curve p^*. The only relevant informations in dual space are topological relations between elements of the complex. We use here any duality relation equivalent to $l : y\cos\theta - x\sin\theta - u = 0 \mapsto l^* : (\theta, u)$, that is, with the following property: If l and l' are two parallel lines such that l' is above l, then in dual space l^* and l'^* are two points with the same x-coordinate and l'^* is above l^*. This property implies that if a point p is below (resp. above) a line l, then the dual point l^* is above (resp. below) the dual

curve p^*.

Figure 1 shows the elements of the complex represented in dual space. This figure also shows the general structure of a

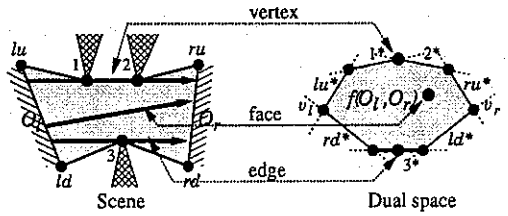


Figure 1: Visibility complex in dual space.

face. A face has two extremal vertices (v_l and v_r) that separate its frontier into two chains of edges: a chain of upper edges (lu^*, 1^*, 2^*, and ru^*) and a chain of lower edges (rd^*, 3^*, and ld^*). The complex has a lower and an upper semi-infinite faces of label (O_∞, O_∞) that represent rays that are outside the convex hull of the scene.

Edges of the complex corresponding to rays passing through p have p^* as supporting curve. As shown in Figure 2, edges whose supporting curve is ru^* (resp. rd^*, lu^*, and ld^*) can be in fact divided into sub-edges: Such edges are then called *fat edges*. They can occur only at the beginning or at the end of chains of edges and will be called *left/right-upper/lower* fat edges.

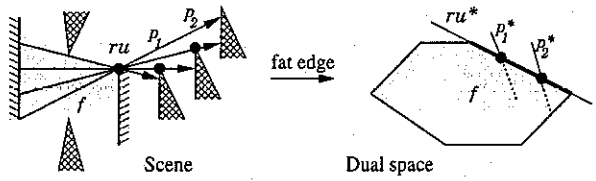


Figure 2: Right-upper fat edges.

The visibility complex of a polygonal scene of n total polygon vertices has a size $O(k)$ — k size of the visibility graph — and can be computed in optimal $O(n \log n + k)$ time and $O(n)$ working space with the algorithm we have described in [Riv97a].

3 Walking in $\mathcal{C}_l(O)$, elements of the complex of label $(O, .)$

The visibility complex encodes all visibility relations between objects of the scene. However, in some applications only a subset of these relations must be used at a time, and visiting all the elements of the complex is then a waste of time. For example, in lighting simulations (see [ORDP96] for more informations on how to use the visibility complex for radiosity computations), only faces of label $(O, .)$ (i.e., O is the left object of their label) must be processed to update the illumination of an object O.

Let $\mathcal{C}_l(O)$ denote the set of elements of the complex of label $(O, .)$ (we do not include in $\mathcal{C}_l(O_\infty)$ the two semi-infinite faces of label (O_∞, O_∞)). To visit all faces of $\mathcal{C}_l(O)$, we define an order on theses faces, and visit them in order.

Let us consider two faces f and f' of $\mathcal{C}_l(O)$ incident to a same edge e. Notice that, since both faces have label $(O, .)$, one face is above e and the other one is below e, that is e is an upper edge of one face and a lower edge of the other. If f is below e, then f' is above e and we say that $f < f'$ (and vice versa). We can extend this order: We say that $f \prec f'$ if there is a sequence (f_i) of faces of $\mathcal{C}_l(O)$ such that $f_0 = f$, $f_k = f'$ and $\forall\, 0 \leq i < k$, $f_i < f_{i+1}$. It can be proved that any two faces f and f' of $\mathcal{C}_l(O)$ are comparable with respect to $\prec$, therefore:

Proposition 1 *The relation $\prec$ on faces of $\mathcal{C}_l(O)$ defined by $f \prec f'$ iff there exists a sequence (f_i) of faces of $\mathcal{C}_l(O)$ such that $f_0 = f$, $f_k = f'$ and $\forall\, 0 \leq i < k$ there exists an edge e_i that is an upper edge of f_i and a lower edge of f_{i+1}, is a total order on faces of $\mathcal{C}_l(O)$.*

Given a face f, we can now define its previous face $prev(f) = max_\prec\{f' \in \mathcal{C}_l(O) \mid f' \prec f\}$ and its next face $next(f) = min_\prec\{f' \in \mathcal{C}_l(O) \mid f \prec f'\}$.

All faces incident to a lower (resp. upper) edge of f are inferior (resp. superior) to f. These inferior faces are of two

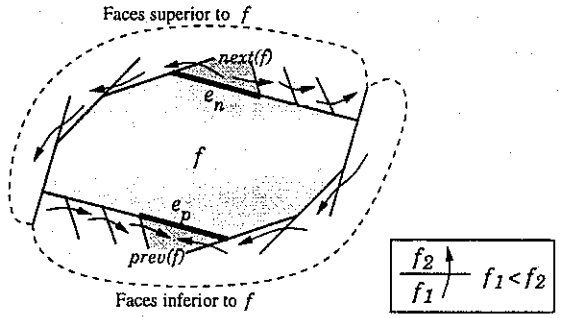


Figure 3: Ordering of faces of $\mathcal{C}_l(O)$.

sorts: faces incident to sub-edges of the left-lower fat edge, that are ordered from left to right, and faces incident to normal lower edges, that are ordered from right to left (figure 3). Therefore, the previous face of f is the face incident to the last sub-edge of the left-lower fat edge of f if this edge exists, incident to the first lower edge else. Similarly, the next face of f is the face incident to the first sub-edge of the right-upper fat edge if this edge exists, incident to the last upper edge else.

A face always has a previous and a next face in $\mathcal{C}_l(O_\infty)$. If the complex is "warped" modulo 2π (i.e., rays of slopes differing by 2π are identified), then by starting from f and always walking into the next (resp. previous) face we visit all faces of $\mathcal{C}_l(O_\infty)$ and "come back" to f. Figure 4 shows an example of walk in $\mathcal{C}_l(O_\infty)$ when faces are visited in decreasing order.

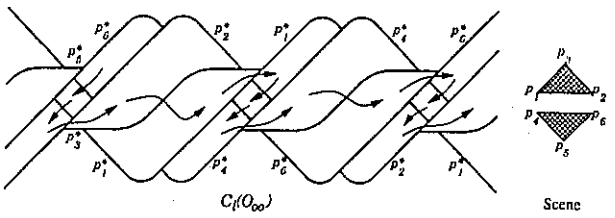


Figure 4: Visiting faces of $C_l(O_\infty)$ in decreasing order.

On the other hand, if O is a line segment (p, q) of the scene, then the previous (resp. next) face of f may not belongs to $C_l(O)$. If f has only one lower edge whose supporting curve is p^*, then $prev(f)$ does not belongs to $C_l(O)$. In this case, the left extremal vertex of f is $v_l = p^* \cap q^*$ and corresponds to the ray (p, q): f is the first face of $C_l(O)$. Likewise, if f has only one upper edge whose supporting curve is q^*, then $next(f)$ does not belongs to $C_l(O)$. The right extremal vertex of f is $v_r = q^* \cap p^*$ and corresponds to the ray (p, q): f is the last face of $C_l(O)$.

So $C_l(O)$ has a finite number of faces that are located between the curves q^* and p^*. To visit all faces of $C_l(O)$, we just start from its first face (which can be found in $O(\log n)$ time if needed) and walk into the next face until encountering the last face.

If each face has a pointor to the first (resp. last) sub-edges of its right (resp. left) fat edges (pointors that can be computed during the construction of the complex), then the previous and the next faces of f can be found in constant time:

Proposition 2 *Without any supplementary data structure, faces of $C_l(O)$ can be visited in increasing (resp. decreasing) order in optimal time proportional to the number of visited faces.*

4 Topologically sweeping vertices of $C_l(O)$

We have seen that we can visit the faces of $C_l(O)$ in optimal time. If for each face f we visit each of its incident vertices, then we can visit all vertices of $C_l(O)$ in optimal time: Such vertices are incident to at most three faces of $C_l(O)$. However, visiting a vertex several time is not practical: If a vertex must be processed only once, then we must keep an historic to check whether a vertex has been visited before. Moreover, there is a natural partial order on the vertices of the complex — $v < v'$ if there is a monotonous path of consecutive edges from v to v' — and it is more interesting to perform a topological sweep of the vertices of $C_l(O)$, that is, to visit then in a way compatible with their order: We can then use the coherence of the sweep to update informations in constant time instead of recomputing them each time.

We use the walk devised in the previous section to do a topological sweep of these vertices in increasing order: We walk on the faces of $C_l(O)$, and for each visited face f we sweep some of its incident vertices.

Those swept vertices must be chosen so that when the walk is completed, (1) each vertex of $C_l(O)$ has been swept exactly once (therefore only a subset of vertices of f must be swept when f is visited), (2) the sweep is coherent locally, that is, for each face f its vertices have been swept from left to right, and (3) the sweep is coherent globally. Property (2) is a consequence of property (3), but we mention it because it helps to devise the sweep.

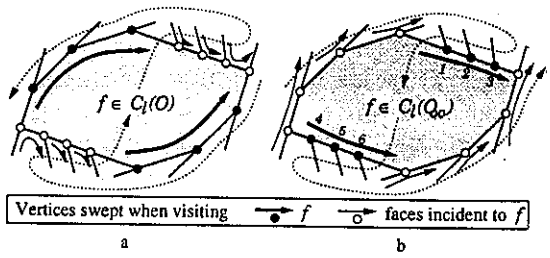


Figure 5: Sweeping vertices in increasing order **a.** in $C_l(O)$, **b.** in $C_l(O_\infty)$.

We first consider $C_l(O)$, with $O \neq O_\infty$. Figure 5(a) shows a face f of $C_l(O)$ and the order in which incident faces of f are visited (dotted arrows) in increasing order. We see that when we visit f, we can sweep neither its extremal vertices, nor vertices that separate sub-edges of its right-upper (resp. left-lower) fat edge: If we did, the local coherence of property (2) would not be satisfied. So we visit only the other incident vertices (black circles, bold arrows), those of upper edges

e = first upper edge of f
  while $e \notin$ right-upper fat edge
    sweep left vertex of e
    e = next upper edge

and those of lower edges

e = last lower edge of f
  while $e \notin$ left-lower fat edge
    sweep right vertex of e
    e = previous upper edge

as shown in figure 5(a).

We see that incident vertices that are not swept when visiting f are swept when visiting incident faces of f during the walk (white circles, normal arrows), and finally each vertex is swept exactly once (with the exception of the left (resp. right) vertex of the first (resp. last) face of the walk, but we just have to sweep them before (resp. after) the walk). We also see that vertices incident to f are swept correctly from left to right. By considering a dual curve and all successive faces incident to this curve, we can show that the sweep is also coherent globally.

Although this sweep is correct for $\mathcal{C}_l(O)$, it cannot be used for $\mathcal{C}_l(O_\infty)$: The face incident to the first (resp. last) subedge of the left-lower (resp. right-upper) fat edge of f may be a semi-infinite face of label (O_∞, O_∞), and since these faces are not visited during the walk, some vertices may not be swept.

So we perform the walk by visiting faces in decreasing order (dotted arrows in figure 5b). We see that this time only vertices subdividing the right-upper and the left-lower fat edges of f can be swept. We sweep them from left to right, by visiting first vertices incident to the right-upper fat edge, then those incident to the left-lower fat edge (black circles, bold arrows). This order is important: A dual curve may cut first the right-upper fat edge of f and cut next the left-lower fat edge of f. Other incident vertices of f are swept when visiting other faces (white circles, normal arrows). As in the previous algorithm, it can be shown that this sweep visits each vertex exactly once and is coherent globally.

This sweep is only valid for $\mathcal{C}_l(O_\infty)$ and cannot be used for $\mathcal{C}_l(O)$ when O is a line segment (p, q) of the scene: The first upper (resp last lower) edge of f may be supported by q^* (resp. p^*), and in this case faces incident to these edges would not belong to $\mathcal{C}_l(O)$ and vertices incident to these edges would not be visited.

Both algorithms have their counterpart for sweeping vertices in decreasing order: The walk is done in reverse order and vertices are swept in decreasing order. Since the walk is done in optimal time,

Proposition 3 *The vertices of $\mathcal{C}_l(O)$ can be swept topologically in increasing (resp. decreasing) order in optimal time proportional to the number of vertices swept.*

Figure 6 shows an exemple of sweep: It takes the walk of figure 4 and shows how vertices are swept during the walk. The order in which vertices are swept is similar to the order

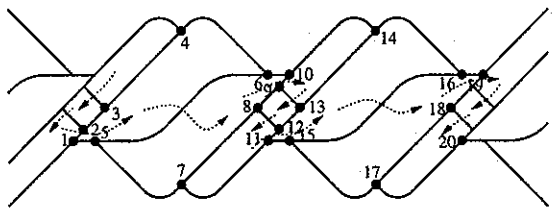


Figure 6: Sweeping vertices of $\mathcal{C}_l(O_\infty)$.

in which vertices of an arrangement of lines are swept with the algorithm of Overmars and Welzl [OW88]. This algorithm, a simplified version of the original algorithm of Edelsbrunner and Guibas [EG86], uses only one horizon tree and sweeps each time the leftmost upper vertex of the upper horizon tree.

Finally, we can notice that other combinations of walk/sweep produce partial sweeps that visit only interior vertices (i.e., vertices incident to three faces of $\mathcal{C}_l(O)$).

5 Walking in the zone of a line segment

We want now to compute the visibility polygon of a line segment $s = (p, q)$, oriented from p to q, which is outside the convex hull of the scene. We must handle the objects of the scene weakly visible from the right side of s.

We must therefore consider faces of the complex that contain rays issued from O_∞ and passing through s (from left to right). The set of rays passing through s is represented in the visibility complex by the zone comprised between the dual curves p^* and q^* of the extremities of s, that is, the zone located below q^* and above p^*. So we consider the subset of elements of the complex of $\mathcal{C}_l(O_\infty)$ that intersect this zone. We note this subset $\mathcal{C}_l(s)$ and call it the *zone of s*.

We can define a total order on faces of $\mathcal{C}_l(s)$ in the same way we did for faces of $\mathcal{C}_l(O)$. However, when searching the next and the previous face of a face f, we must now be carefull to stay in $\mathcal{C}_l(s)$: The next (resp. previous) face of f in $\mathcal{C}_l(O)$ may not belongs to $\mathcal{C}_l(s)$. To compute the upper edge e_{ns} incident to the next face $next_s(f)$ of f in $\mathcal{C}_l(s)$, we must do some checking (figure 7a):

e_n = edge incident to $next(f)$ in $\mathcal{C}_l(O)$
if e_n is above q^*
  then e_{ns} = upper edge of f cut by q^*
else if e_n is below p^*
  then e_{ns} = upper edge of f cut by p^*
else $e_{ns} = e_n$

We compute the lower edge e_{ps} incident to the previous face $prev_s(f)$ of f in $\mathcal{C}_l(s)$ similarly (figure 7b):

e_p = edge incident to $prev(f)$ in $\mathcal{C}_l(O)$
if e_p is above q^*
  then e_{ps} = lower edge of f cut by q^*
else if e_p is below p^*
  then e_{ps} = lower edge of f cut by p^*
else $e_{ps} = e_p$

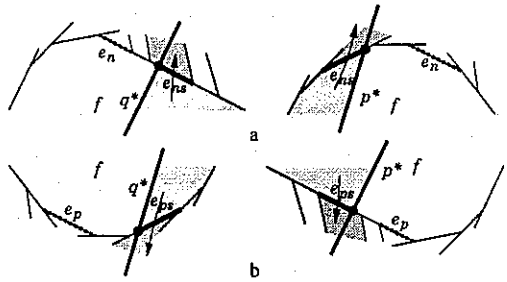


Figure 7: **a.** Computing $next_s(f)$. **b.** Computing $prev_s(f)$.

$\mathcal{C}_l(s)$ has a finite number of faces, and therefore has a first and a last face. More precisely, a face f does not have a next (resp. previous) face in $\mathcal{C}_l(s)$ either if it contains the vertex $v = p^* \cap q^*$ corresponding to the ray (p, q) (resp.

(q,p)), or if its next (resp. previous) face is a semi-infinite face (O_∞, O_∞).

If the supporting line of s does not cut the convex hull of the scene, then $v = p^* \cap q^*$ is in a semi-infinite face (O_∞, O_∞). In this case, the first face of $C_l(s)$ is the face incident to the first edge cut by p^*, and the last face is the face incident to the last edge cut by q^* (figure 8 left). Both faces are incident to a semi-infinite face (O_∞, O_∞).

If the oriented supporting line (p,q) of s cuts the convex hull of the scene such that s is behind the scene, then the first face of $C_l(s)$ is the face incident to the edge cut by p^* and to the lower semi-infinite face (O_∞, O_∞), and the last face is the face containing $v = p^* \cap q^*$ (figure 8 middle).

If the line (p,q) cuts the convex hull of the scene such that s is before the scene, then the first face of $C_l(s)$ is the face containing $v = p^* \cap q^*$, and the last face is the face incident to the edge cut by q^* and incident to the upper semi-infinite face (O_∞, O_∞) (figure 8 right).

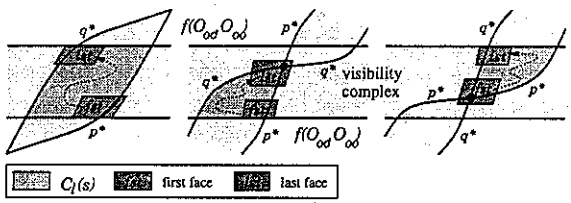


Figure 8: First and last faces of the zone of s.

Checking if e_n (resp. e_p) is below p^* or above q^* is done in constant time. The computation of all the edges cut by p^* (resp. q^*) is in fact the computation of the view around p (resp. q). These views can be computed with a sweep algorithm and need not be computed in advance: The edges cut by p^* (resp. q^*) can be computed one at a time when needed so that the computation of the walk and of the views are synchronized. So the sweep can be performed with no additional data structure, and

Proposition 4 *Given a line segment $s = (p,q)$ outside the convex hull of the scene, the n_f faces of $C_l(s)$ can be visited in increasing (resp. decreasing) order in $O(\log n + n_f + t_v(p) + t_v(q))$ time, where $t_v(p)$ (resp. $t_v(q)$) is the time to compute the view around p (resp. q).*

The view around a point can be computed by two sweep algorithms respectively in $O(v \log n)$ time, v size of the view, and in $\Omega(v)$ and $O(nv)$ time where these bounds are tight (see [PV96] and [Riv97a]).

We show in the remaining sections how the walk in the zone of s can be used to compute the visibility polygon of s, and to maintain the view around a point moving from p to q.

6 Computing the visibility polygon of a line segment

The visibility polygon of a line segment $s = (p,q)$ is the set of points of objects of the scene that are visible from (at least) one point in s. Every two side of the polygon is a transversal side supported by an object of the scene. The other sides are radial sides that link transversal sides (figure 9). We consider here that s is outside the convex hull of the scene.

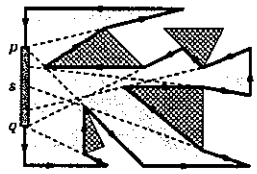


Figure 9: Visibility polygon of a line segment.

We compute the visibility polygon by sweeping its successive transversal sides. A transversal side s_1 of the visibility polygon supported by an object O_1 is the set of right extremities of rays of a face f_1, face of $C_l(s)$ of label (O_∞, O_1). s_1 can be considered as the portion of O_1 lightened by the neon s, the rays lightening s_1 being those of f_1.

Let $f2$ be the next face of f_1 in $C_l(s)$ ($f_2 = next_s(f_1)$), and let (O_∞, O_2) be its label. Then it can be shown that the next transversal side of the visibility polygon is the part of O_2 "lightened" by rays of f_2. Moreover, if e denotes the edge incident to f_1 and f_2, then the radial side linking s_1 to s_2 (when it is not reduced to a point) is supported by the ray whose dual point in the complex is either a vertex extremity of e, or the intersection point of e and p^* (resp. q^*).

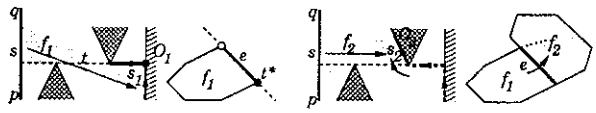


Figure 10: Sweep of the visibility polygon.

The walk in $C_l(s)$ allows us to compute the visibility polygon of s easily:

Proposition 5 *The visibility polygon of $s = (p,q)$ can be computed in $O(vis(s) + t_v(p) + t_v(q))$ time, where $vis(s)$ is the size of the visibility polygon and $t_v(p)$ (resp. $t_v(q)$) the time to compute the view around p (resp. q).*

Although its complexity is low, this algorithm is only close to optimal: Some points seen by p (resp. q) may not correspond to a radial side of the visibility polygon. Notice also that the visibility polygon is not necessarily a simple polygon.

7 Maintaining the view around a moving point

We show in this section how to maintain the view around a point moving from p to q along a line segment $s = (p, q)$ which is outside the convex hull of the scene.

In [Riv97b] we have proposed two algorithms that maintain the view around p_v by processing visibility changes in their order of occurence. Here we use the walk in $C_l(s)$ to process visibility events in topological order.

The view around a moving point p_v changes when p_v crosses an (extended) edge of the visibility graph. In the visibiliy complex, the view around p_v is the set of consecutive edges cut by the dual curve p_v^*. The view changes when the dual curve p_v^* sweeps a vertex of the complex. The vertices swept by p_v^* during a displacement of p_v must be processed in topological order so that the view around p_v can be updated in constant time at each visibility change.

We show here how to maintain the view when the supporting line of s does no intersect the convex hull of the scene (others cases are similar). After computing the view around p, we must sweep vertices of $C(s)$ from p^* to q^*, that is in decreasing order. When we are in a face f, instead of computing directly the edge incident to the next face, we use the sweep of the vertices to find the next face. We first sweep sub-edges of the left-lower fat edge in decreasing order. If p^* cuts the fat edge, we start from the sub-edge cut by p^*, else we start from the last sub-edge of the fat edge. We sweep the previous sub-edges until encountering a sub-edge cut by q^* or the first sub-edge of the fat edge. Then we sweep the right-upper fat edge the same way, and the last sub-edge swept is the edge incident to the next face.

With this method, the view around q is not computed directly, but computed implicitly after all the updates of the view around p.

Proposition 6 *Let p_v be a point moving from p to q along the line segment $s = (p, q)$. After computing the view around p, the view around p_v can be maintained in total $O(\max(v(p), v(p, q)))$ time, where $v(p)$ is the size of the view around p and $v(p, q)$ the number of changes of visibility along s.*

We have previously presented in [Riv97b] two algorithms for maintaining the view around a point. The algorithm of this paper has the advantage over these two algorithms of updating the view in constant time (instead of $O(\log^2 v(p_v))$ (resp. $O(\log n)$) time) at each change of visibility. Moreover, it does not need data structures such as a dynamic convex hull or a priority queue and is in fact independant of the real trajectory of p_v between p and q.

If a program does not need to process visibility changes in temporal order, but only in topological order, then this algorithm improves the running time of the second algorithm in [Riv97b] (which needs an initialization in $O(v(p) \log v(p))$ time).

When used for consecutive moves, this algorithm has however the disadvantage over the first algorithm in [Riv97b] of visiting all faces of $C(s)$, even if some of these faces do not yield a visibility change, and therefore runs in at least $O(v)$ time for each move.

8 Conclusion

We have shown how to sweep only some parts of the visibility complex in optimal time and without any additional data structure. Although we have studied in this paper walks in subsets of elements of the complex having the same left object in their label, all the algorithms have their counterpart for subsets of elements having the same right object in their label. All these algorithms appear to be relatively simple and have been or are being implemented.

We have also shown two applications of these walks: Computing the visibility polygon of a line segment and maintaining the view around a moving point.

Asano, Guibas, and Tokuyama [AGT94] have shown how to sweep some parts of an arrangement of lines in the plane without the arrangement being built. It would be interesting to see if our walks in the complex can also be performed without the complex being built.

We are also studying if these walk could be used in algorithms for maintaining the visibility complex upon insertion or deletion of polygons in a scene, to identify the elements of the complex that are modified by this insertion/deletion.

Finally, we have extended the algorithm for computing the visibility polygon of a line segment to line segments inside the scene. However work remains to be done to ameliorate its complexity.

References

[AGT94] Te. Asano, L. J. Guibas, and T. Tokuyama. Walking on an arrangement topologically. *Internat. J. Comput. Geom. Appl.*, 4:123–151, 1994.

[EG86] H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. In *Proc. 18th Annu. ACM Sympos. Theory Comput.*, pages 389–403, 1986.

[ORDP96] R. Orti, S. Rivière, F. Durand, and C. Puech. Radiosity for dynamic scenes in flatland with the visibility complex. *Comput. Graph. Forum*, 15(3):237–248, 1996. Proc. Eurographics '96.

[OW88] M. H. Overmars and E. Welzl. New methods for computing visibility graphs. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 164–171, 1988.

[PV96] M. Pocchiola and G. Vegter. The visibility complex. *Internat. J. Comput. Geom. Appl.*, 6(3):279–308, 1996. Special issue devoted to ACM-SoCG'93.

[Riv97a] S. Rivière. *Visibility computations in 2D polygonal scenes.* Ph.D. thesis, Université Joseph Fourier, Grenoble, France, 1997.

[Riv97b] Stéphane Rivière. Dynamic visibility in polygonal scenes with the visibility complex. In *Proc. 13th Annu. ACM Sympos. Comput. Geom. (Communication)*, 1997.

The 3D Visibility Complex: a unified data-structure for global visibility of scenes of polygons and smooth objects

Frédo Durand, George Drettakis and Claude Puech
*i*MAGIS-GRAVIR/INRIA *

Abstract

In this paper we describe a unified data-structure, the *3D Visibility Complex* which encodes the visibility information of a 3D scene of polygons and smooth convex objects. This data-structure is a partition of the maximal free segments and is based on the characterization of the topological changes of visibility along critical line sets. We show that the size k of the complex is $\Omega(n)$ and $O(n^4)$ and we give an output sensitive algorithm to build it in time $O((n^3 + k)\log n)$.

This theoretical work has already been used to define a practical data-structure, the *Visibility Skeleton* described in a companion paper.

1 Introduction

Visibility is a crucial issue; motion planning in robotics, object recognition in computer vision, lighting simulation or view maintenance in computer graphics are some examples where global visibility computations are required. The notion of "coherence" is often cited as the key to treat these problems efficiently and not restart every computation from scratch, but its characterization is not straightforward.

The usual space-subdivision methods do not translate the line nature of visibility, since a line of sight intersects many cells of any subdivision.

Computational geometers have characterized sets of lines in space by using Plücker duality. It is an oriented projective 5D dual space in which lines of space are naturally and linearly embedded (lines intersecting a given line are associated with hyperplanes). Its main drawback is the necessity of an intersection with the Plücker hypersurface [CEG+96, Pel90]. The scenes considered have always been polygonal and are mainly restricted to isothetic or c-oriented polygons. (In fact there exists a few results on ray-shooting with spheres involving parametric search without Plücker coordinates [MS97]). These techniques have been used by Teller in computer graphics to compute the antipenumbra cast by an area light source through polygonal portals [Tel92]. The problem with these methods is that intersections of lines with the entire scene are considered; occlusion is not really treated.

In computer vision, the *aspect graph* has been developed to characterize the viewpoints from which the scene has the same topological aspect. The viewing space (S^2 for orthographic projection, R^3 for perspective projection) is partitioned along *visual events*. Construction algorithms have been developed for polygons and algebraic objects, both for orthographic and perspective projection, and some of them have been implemented; see [EBD92] for a good survey. A main drawback of aspect graphs is their size: $O(n^6)$ for orthographic projection, and $O(n^9)$ for perspective projection.

To build the aspect graph, Plantinga and Dyer [PD90] defined an intermediate data structure called the *asp*. For the orthographic case, it is a partition of the 4D space of oriented lines of space according to the first object they hit, and for the perspective case it is a partition of the 5D space of oriented half lines (rays). This approach has been limited to polygonal scenes. It was applied to maintain views, but the degrees of freedom allowed by the implementation were limited to rotation along a predefined axis [PDS90].

In lighting simulation, researchers have computed the discontinuities of the lighting function (which correspond to the limits of umbra and penumbra) also called *discontinuity meshes*. This characterizes the visibility of a light source. Initially only a subset of discontinuities where computed (e.g., [LTG93]), followed by algorithms computing all the discontinuities, together with a structure, the *backprojection*, which encodes the topological aspect of the light source[DF94, SG94]. These approaches are nonetheless restricted to a single light-source at a time.

Recently, a data-structure which encodes all the visibility information of a 2D scene called the *Visibility Complex* has been defined [PV96]. This structure is a partition of the set of maximal free segments according to the object they touch. Optimal construction algorithms have been developed for smooth convex objects [PV96] as well as polygons [Riv97] and used for lighting simulation [ORDP96].

In [DDP96] we introduced the *3D Visibility Complex* for

* Laboratoire GRAVIR / IMAG. iMAGIS is a joint research project of CNRS/INRIA/INPG/UJF. Postal address: B.P. 53, F-38041 Grenoble Cedex 9, France. Contact E-mail: Frederic.Durand@imag.fr. http://www-imagis.imag.fr

scenes of convex smooth objects (the polygonal case was simply mentioned). An $O(n^4 \log n)$ brute-force algorithm was roughly sketched, and applications for lighting simulation, walkthroughs and aspect graph computation were proposed.

In this paper, we present a unified version of the 3D visibility complex for scenes of polygons and smooth convex objects. It is based on a complete catalogue of critical line sets which are lines where visibility changes. We derive bounds for the size of the complex and present an output sensitive construction algorithm.

Moreover, the formalism described in this article has been used to develop and implement a global visibility datastructure called the *Visibility Skeleton* [DDP97]. It is a simplified version of the 3D visibility complex for polygonal scenes built using a brute-force algorithm.

2 Scenes and maximal free segments

We consider scenes of polygons and algebraic smooth convex objects. Concave objects and piecewise smooth objects are beyond the scope of this article but could be handled by considering other critical line sets described by the theory of singularity [PPK92, Rie87]. The algebraic objects are assumed to have bounded degree. In what follows, n represents the overall complexity of the scene which is the total number of objects, polygons, edges and vertices. The objects are assumed to be in general position; degeneracy issues are not addressed in this paper.

In this work we do not consider lines but maximal free segments to take occlusion efficiently into account. Intuitively, a segment represents a class of rays, and we want to group the rays that "see" the same objects. Since many segments can be collinear, we need a fifth dimension to distinguish them. But it is not a continuous dimension: there is only a finite number of segments collinear to one line. See figure 1(a) where a 2d equivalent is shown. The segments a and b are collinear, t is tangent to the object and is adjacent to segments above and below the object. Topologically we have a branching structure represented in fig. 1 for parallel segments. Note that almost everywhere the graph is locally 1-dimensional. Similarly in 3D, the segment space is a 4D space embedded in 5D. This can be seen as a unification of the spaces used by Plantinga and Dyer [PD90]: in the orthographic case they deal with a 4D space and in the perspective case with a 5D space.

We use the same parameterization for lines as [PD90, DDP96]: they are represented by two coordinates of direction, the angles θ (azimuth) and φ (elevation) which are the spherical coordinates of the director vector, and the coordinates (u, v) of the projection onto the plane perpendicular to the line and going through the origin (the axes of of the plane are chosen such as u is orthogonal to both the director vector and the vertical). See figure 1(b). Note that if φ is fixed we obtain all the lines contained in a set of parallel planes.

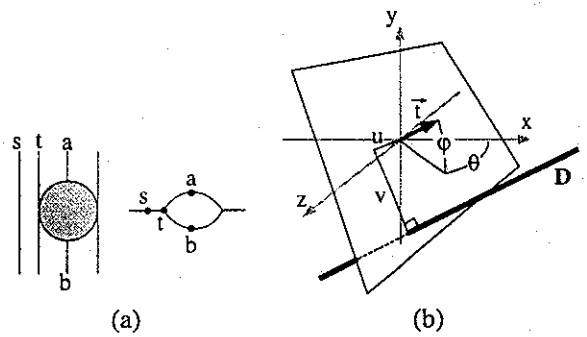


Figure 1: (a) 2D equivalent of the segment space: Parallel segments in the scene and local topology with branchings. (b) Parameterization of lines in space.

We call it a φ-slice. If we also fix v, we obtain the lines of a plane in which θ and u are the polar coordinates. We call it a φv-slice.

3 Critical segments

We define a segment to be in general position if it touches objects only at its extremities. A segment that touches objects in its interior will be called *critical*. At such an intersection there is a *local event*. If a segment touches more than one object in its interior, we call this a *multilocal event*. Critical segments are grouped into *critical segment sets*. The dimension of such a set can be seen as the number of degrees of freedom a segment has to keep the events. We can also refer to the codimension of such a set, which is the complement to the dimension of the space (the number of fixed degrees of freedom).

For the class of scenes we consider, there are two kinds of local events: tangency events and vertex events. The object or the vertex are called the *generators* of the event. To stay tangent to an object, a segment has three degrees of freedom. It is of codimension 1. It is of course the same when a segment goes through the edge of a polygon. We call this a T event from tangency (also referred to as E from edge in the aspect graph or discontinuity meshing literature which deals with polygonal scenes). A segment that goes through a vertex has two degrees of freedom (rotation), and thus has codimension 2. We call it a V event.

The combination of many local events causes a multilocal event, and the codimensions are added. We use the notation $+$ to describe such a combination. For example, a segment that is tangent to an object and that goes through a vertex belongs to a $T + V$ critical line set of codimension $1 + 2 = 3$ (it is a 1D set).

There is also a different kind of multilocal event that was not described in [DDP96]. A segment can be tangent to two objects and belong to one of their common tangent planes. In this case, the common tangent plane adds one codimension and we use the notation $++$. For example $T + +T$ critical

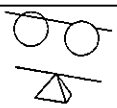
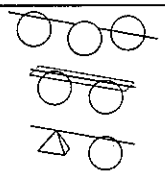
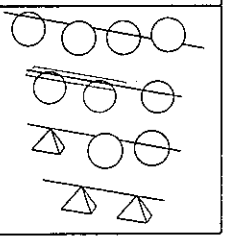
Dimension	Type	Configuration
3	T	
2	T+T V	
1	T+T+T T++T T+V	
0	T+T+T+T T++T+T T+T+V V+V	

Table 1: faces of the visibility complex.

segment sets have codimension $1+1+1=3$ (1D set). (One may think of the example of two parallel cylinders and notice that lines contained in a bitangent plane have two degrees of freedom. This case is not considered here because it is degenerated.) These events are crucial for dynamic maintenance of views, aspect graphs and discontinuity meshes. For example a sphere hidden behind another sphere will appear when their outlines are tangent, that is when the viewpoint lies on a $T++T$ segment.

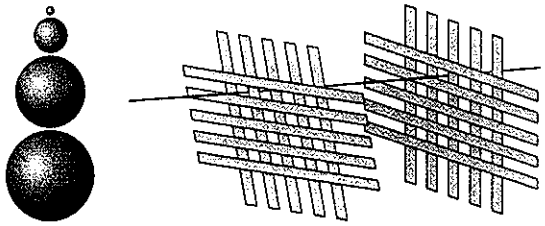
Each local event corresponds to an algebraic equation: a line tangent to an algebraic object or going through a vertex. A set of critical segments can thus be associated with the connected set of lines verifying the corresponding set of equations.

Events caused by faces are considered as $T+T$ events since they involve two edges. In the same way, segments going through an edge are $V+V$ events. The reason why the case of vertices (which could be seen as two edges events) is distinguished is that they introduce "discontinuities" at the end of edges and require a specific treatment as we shall see in section 5.4.

4 The 3D Visibility Complex

The *3D visibility complex* is the partition of maximal free segments of 3-space into connected components according to the objects they touch. Its faces of dimension 4 are maximal connected components of segments in general position with the two same objects at their extremities.

The different faces of lower dimension correspond to critical segments as summarized in table 1.



(a) (b)

Figure 2: (a) Scene with an $O(n)$ Visibility Complex (b) Scene with an $O(n^4)$ Visibility Complex (an example of $T+T+T+T$ critical line is shown).

Theorem 1 *The size of the 3D visibility complex is $\Omega(n)$ and $O(n^4)$ where n is the complexity of the scene.*

Proof (sketched)

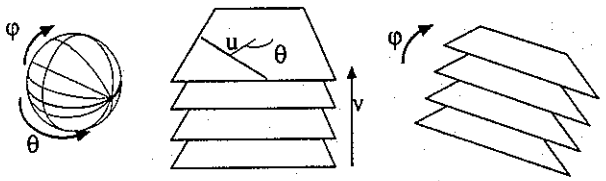
The number of $(k+1)$-faces adjacent to a k-face is bounded. For example a 1-face $T_1+T_2+T_3$ is adjacent to five 2-faces: two faces T_1+T_2 (there are two different faces because one extremity of the segments can lie on the object tangent at T_2 or not. See [DDP96]), T_1+T_3 and two T_2+T_3.

Each 4-face is adjacent to at least one 3-face, a 3-face to at least one 2-face, and a 2-face to at least one 1-face. We just sketch the demonstration. For a given face F of the complex, we consider the associated critical line set S. This set of lines contains a line set S' with one more codimension (one of the lines tangent to one object is also tangent to a second object, one of the lines tangent to two objects belongs to one of their common tangent plane, and one line going through a vertex is tangent to an object). Consider a continuous path from the line associated with a segment s of F to one of S', and the corresponding continuous path over the segments. If all the segments of this path have the same extremities, F is adjacent to the face with one more codimension associated with S, otherwise when the extremity changes there is a tangency local event and one more codimension.

Note that a 1-face may be adjacent to no 0-face (we give an example below of a scene without a 0-face).

So the size of the complex is bounded by the number of 1-faces which are not adjacent to a 0-face plus the number of 0-faces. For each kind of events, the number of possible systems of algebraic equations depends on the number of objects implicated, the $T+T+T+T$ critical line sets are thus the most numerous with $O(n^4)$.

We show in figure 2(a) an example of a scene with a visibility complex of size $O(n)$: there is one $T++T$ face for each pair of neighbour spheres. Note there is no 0-face in that case. The scene in figure 2(b) is the same as in [PD90] and has an $O(n^4)$ visibility complex. There are two "grids", each one composed of two very slightly distant orthogonal sets of $\frac{n}{4}$ parallel rectangles (this is also valid with thin ellipsoids). Consider a rectangle in each of the four sets: there is always a $T+T+T+T$ critical segment.



(a) (b (c)

Figure 3: (a) Parameterization of the directions. (b) Initial v sweep (c) φ sweep.

Visual events considered in the aspect graph literature [EBD92, PD90] correspond to the 1-faces of the visibility complex. For example the topology of a view changes when a vertex and an edge are aligned from the viewpoint. The aspect graph is in fact the arrangement of those events in the viewing space. This explains its size: $O(n^6)$ in the orthographic case where the viewing space is S^2 and $O(n^9)$ in the perspective case where the viewing space is R^3.

In [DDP97] we presented a data-structure called the *Visibility Skeleton* which corresponds to the graph of the 0 and 1-faces of the visibility complex. First experiments with a few typical computer-graphics scenes show that the number of these faces (and thus the size of the complex) is about quadratic in the number of input polygons.

5 Output-sensitive sweep

Our algorithm is a double sweep with a preprocessing phase. First the scene is swept by a horizontal plane and a 2D Visibility Complex [PV96] of the φv-slice is maintained (figure 3(b)). We then sweep φ (figure 3(c)), but some 0-faces can not be detected during this sweep and have to be preprocessed.

5.1 Sweeping the initial slice

To build the initial φ-slice, we first maintain a φv-slice of the 3D visibility complex which corresponds to the 2D visibility complex [PV96] of the sweeping plane. We briefly review the 2D visibility complex. It is the partition of the segments of the planes according to the objects they touch. Its 2D faces are connected components of segments touching the same objects (they are φv-slices of the 4-faces of the 3D visibility complex). They are bounded by edges which correspond to segments tangent to one object (φv slices of the 3-faces T) and vertices which are free bitangents of the 2D scene (φv-slices of 2-faces $T+T$). Since a view around a point corresponds to the extremities of the segments going through this point, it corresponds to the traversal of the 2D visibility complex along the 1D path of these segments. The object seen changes when the path traverses a new face, which occurs at an edge of the 2D complex. In the case of a polygon, the chain of edges of the 2D complex going through one of its vertices is the view around this vertex.

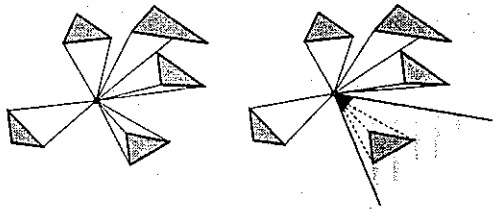


Figure 4: When the first vertex of a polyhedron is swept, the 2D view is computed in the sweeping plane and is restricted for each edge adjacent to the vertex by considering the angle formed by the direction of the two adjacent polygons.

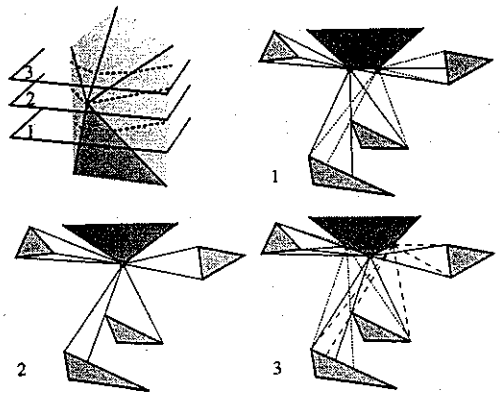


Figure 5: Fusion-restriction of a view around edges when a vertex is swept

The 2D visibility complex has to be updated when the sweeping plane is tangent to an object or contains a vertex and when three 2D slices of objects share a tangent.

When the sweeping plane starts intersecting an object, we have to "insert" this object in our 2D complex. This is done by computing a view around the point of tangency or around the vertex using the current 2D visibility complex. This can be done in $O(v \log n)$ where v is the size of the view using the techniques described in [Riv97]. When the path of this view crosses an edge of the 2D complex it corresponds to a new $T+T$ or $V+T$ face of our 3D complex. In the case of the first vertex of a polygon, the view has to be restricted for each edge of the polyhedron, corresponding to the view seen by a vertex of the 2D slice (see figure 4).

Symmetrically, when an object stops intersecting the sweeping plane, the corresponding faces of the 2D visibility complex are collapsed. These faces are those along the chains of edges corresponding to segments tangent to this object. Their removal can be done in $O(v)$ where v is again the size of the view.

When a vertex in the middle of a polyhedron is encountered the 2D views around the points corresponding to the edges under the vertex have to be merged, and then the view around this vertex has to be restricted for each edge above the vertex, in the same manner as first vertex sweep-events, see figure 5. Each operation is linear in the size of each view.

As the plane moves, three slices of objects can share a tan-

gent (corresponding to a $T+T+T$ face of the 3D complex), in which case the 2D visibility complex is updated using the technique of [Riv97]. Basically, for each bitangent we compute the value of v where it will become tangent to a third object and store these sweep-events in our queue which requires time $O(\log n)$ whenever a bitangent is created.

Finally, a bitangent of the 2D complex can correspond to a common tangent plane. For each bitangent, we compute the value of v for which it will lie on a bitangent plane and insert this sweep-event in the queue. Of course, these sweep-events have to be discarded if the bitangent is collapsed before.

5.2 Principle of the φ sweep

We now have computed a φ-slice of the 3D visibility complex. It is the partition of the segments contained in the set of horizontal planes. In this φ-slice, 1-faces of the complex have dimension 0, 2-faces have dimension 1, and so on.

During the φ-sweep (fig. 8(c)) we maintain this φ-slice as well as a priority queue of sweep-events. In what follows, we will only describe the update of the 1-faces of the visibility complex, the update of the upper dimensional is done at each sweep-event using a catalogue of adjacencies of the 1-faces which for reason of place cannot be given here. As stated before, the number of adjacent upper-dimensional faces is bounded; their update does not affect the complexity.

We first prove that some sweep-events are regular: a 1D component of the φ-slice is collapsed as its two extremities merge. These sweep-events can be detected by computed for each 1D component of the φ-slice the value of φ for which it will collapse. We will then study the case of irregular sweep-events.

5.3 Regular 0-faces

Consider a $T_1 + T_2 + T_3 + T_4$ segments with extremities O_0 and O_5 and elevation angle φ_0 (fig. 6). Consider the $1D$ critical line set $T_1 + T_2 + T_3$. We locally parameterize it by φ and call it $l(\varphi)$. The ruled surface described by $l(\varphi)$ cuts O_4 at φ_0. Two 1-faces of the complex are associated with $l(\varphi)$, one for $\varphi < \varphi_0$ and one for $\varphi > \varphi_0$; one has O_5 at its extremity, the other O_4. It is the same for $T_2 + T_3 + T_4$. Moreover the two 1-faces before φ_0 are adjacent to a 2-face $T_2 + T_3$. In the φ-slice, this 2-face is a 1D set bounded by the slices of $T_1 + T_2 + T_3$ and $T_2 + T_3 + T_4$. This 1D set collapses at φ_0, it is thus a regular sweep-event. It can be detected by considering the adjacent $T+T+T$ faces in the φ-slice and maintaining a priority queue.

The $T + +T + T$ faces can be handled the same way because they are adjacent to a pair of $T + +T$ and a pair of $T + T + T$ 1-faces, and the faces of a pair are associated with the same line set.

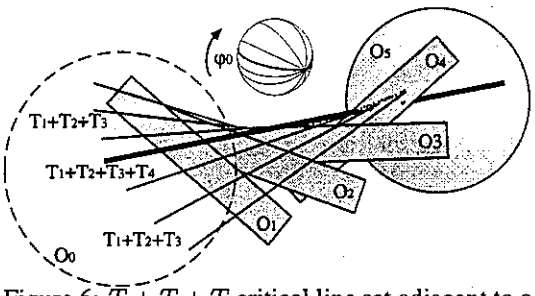


Figure 6: $T + T + T$ critical line set adjadent to a $T + T + T + T$ critical line.

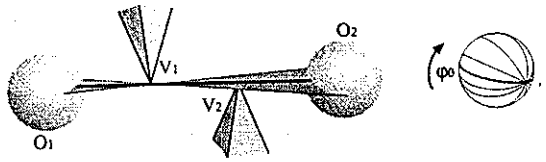


Figure 7: None of the $T + V$ critical segment sets adjacent to this $V + V$ critical segment exist before φ_0

5.4 Irregular 0-faces

Unfortunately, all the 0-faces are not regular sweep-events. The $T + T + V$ and $V + V$ events cannot be detected in this way. The main reason is that vertices represent discontinuities at the end of edges, and we have no guarantee that a 1-face adjacent to such a 0-face exists for $\varphi < \varphi_0$. See figure 7 where the four $T + V$ faces appear at φ_0; this corresponds in the dual space to situation (b) of fig. 8.

These events thus have to be preprocessed by considering all the VV pairs and all the Object-Object-V triplets.

Fortunately, at least one slice of an adjacent 2-face exists before such 0-faces appear (face V_1 in fig 7). The proof is omitted from this version. This face is found using a search structure over the 1D components of the φ-slice ordered by their generators. The 0-face is then tested for occlusion: we test if the generators (V_2 here) lies between the extremities (O_1 and O_2) of the 2-face. It can then be inserted.

5.5 Non monotonic 1-faces

There is another kind of irregular sweep-event. A 1-face of the complex can appear during the sweep without a 0-face event. This is obviously the case for $T + +T$ events since they can be adjacent to no 0-face, but this can also be the case for $T + T + T$ events. Consider the associated line set, it is not necessarily monotonic with respect to φ (see fig. 8(c)). These sweep-events also have to be preprocessed and inserted in the φ-slice with a search over the 1D components.

5.6 Complexity of the algorithm

Theorem 2 *The visibility complex can be built in time $O((k + n^3)\log n)$ where n is the complexity of the scene, and k the number of 0-faces of the complex.*

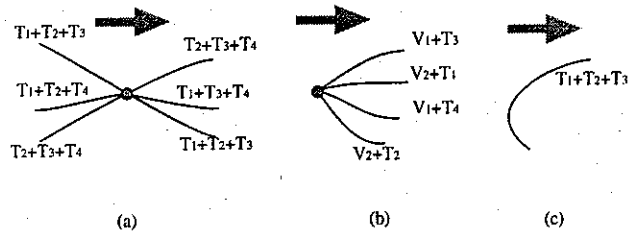


Figure 8: Different sweep-events represented in the dual space. The $T+T+T+T$ event (a) is regular, but the $V+V$ event (b) has to be preprocessed as well as the null derivative with respect to φ of the $T+T+T$ events (c).

During the initial v sweep, each view computation requires time $O(v \log n)$ where v is the size of the view. A view corresponds to the number of 3-faces of the 3d visibility complex adjacent to the appearing/disappearing 2 faces. The total cost is thus bounded by $O(k \log n)$. Each tritangent event requires time $O(\log n)$, here again the cost is bounded by $O(k \log n)$.

During the φ sweep, each regular event requires $O(\log n)$ to maintain the priority queue.

The preprocessing of the other 0-faces and non-monotonic 1-faces requires the enumeration of all the triplets of objects and the insertion of the computed faces in the priority queue, it is therefore $O(n^3 \log n)$.

The output-sensitive nature of this algorithm is very important since experiments on a few polygonal scenes [DDP97] have shown that the number of $T+T+T+T$ segments which is responsible of the theoretical $O(n^4)$ is in fact much less than the number of $T+T+V$ segments.

6 Conclusions and future work

We have introduced a unified data structure, the 3D visibility complex, which encodes the global visibility informations for 3D scenes of polygons and convex smooth objects. Its size k is $\Omega(n)$ and $O(n^4)$ and we have presented an output-sensitive algorithm to build the structure in time $O((n^3 + k) \log n)$.

Future work includes the use of the visibility complex to maintain views around a moving viewpoint, a study of the events involved by concave and piecewise smooth objects, the development of a better construction algorithm, and the incremental update of the visibility complex when an object is moved, added or removed.

Acknowledgments

The authors would like to thank Seth Teller, Michel Pocchiola and Sylvain Petitjean for very fruitful and inspiring discussions.

References

[CEG+96] B. Chazelle, H. Edelsbrunner, L. J. Guibas, M. Sharir, and J. Stolfi. Lines in space: combinatorics and algorithms. *Algorithmica*, 15:428–447, 1996.

[DDP96] F. Durand, G. Drettakis, and C. Puech. The 3d visibility complex, a new approach to the problems of accurate visibility. In *Proc. of 7th Eurographics Workshop on Rendering in Porto, Portugal*, June 1996.

[DDP97] F. Durand, G. Drettakis, and C. Puech. The visibility skeleton: A powerful and efficient multi-purpose global visibility tool. *Computer Graphics (Siggraph'97 Proceedings)*, 1997.

[DF94] G. Drettakis and E. Fiume. A fast shadow algorithm for area light sources using back projection. In *Computer Graphics Proceedings, Annual Conference Series:* SIGGRAPH '94 (Orlando, FL), July 1994.

[EBD92] D. Eggert, K. Bowyer, and C. Dyer. Aspect graphs: State-of-the-art and applications in digital photogrammetry. In *Proceedings of the 17th Congress of the Int. Society for Photogrammetry and Remote Sensing*, 1992.

[LTG93] D. Lischinski, F. Tampieri, and D. Greenberg. Combining hierarchical radiosity and discontinuity meshing. In *Computer Graphics Proceedings, Annual Conference Series:* SIGGRAPH '93 (Anaheim, CA, USA), August 1993.

[MS97] S. Mohaban and M. Sharir. Ray shooting amidst spheres in 3 dimensions and related problems. *to appear in SIAM J. Computing*, 1997.

[ORDP96] R. Orti, S. Rivière, F. Durand, and C. Puech. Radiosity for dynamic scenes in flatland with the visibility complex. In *Proc. of Eurographics*, Poitiers, France, 1996.

[PD90] H. Plantinga and C. R. Dyer. Visibility, occlusion, and the aspect graph. *IJCV*, 1990.

[PDS90] H. Plantinga, C. R. Dyer, and B. Seales. Real-time hidden-line elimination for a rotating polyhedral scene using the aspect representation. In *Proceedings of Graphics Interface '90*, 1990.

[Pel90] M. Pellegrini. Stabbing and ray shooting in 3-dimensional space. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, 1990.

[PPK92] S. Petitjean, J. Ponce, and D.J. Kriegman. Computing exact aspect graphs of curved objects: Algebraic surfaces. *IJCV*, 1992.

[PV96] M. Pocchiola and G. Vegter. Topologically sweeping visibility complexes via pseudo-triangulations. *Discrete Comput. Geom.*, December 1996. special issue devoted to ACM-SoCG'95.

[PV96] M. Pocchiola and G. Vegter. The visibility complex. *Internat. J. Comput. Geom. Appl.*, 96. special issue devoted to ACM-SoCG'93.

[Rie87] J.H. Rieger. On the classification of views of piecewise smooth objects. *Image and Vision Computing*, 1987.

[Riv97] S. Rivière. Dynamic visibility in polygonal scenes with the visibility complex. In *Proc. 13th Annu. ACM Sympos. Computat. Geom.*, 1997.

[SG94] J. Stewart and S. Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 1994)*, Computer Graphics Proceedings, July 1994.

[Tel92] S. Teller. Computing the antipenumbra of an area light source. *Computer Graphics*, July 1992. Proceedings of SIGGRAPH '92 in Chicago (USA).

The width of a convex set on the sphere

F.J. Cobos J.C. Dana C.I. Grima A. Márquez
Departamento de Matemática Aplicada I
Universidad de Sevilla (Spain)
e-mail:{cobos|dana|grima|almar}@cica.es
Fax: +34-5-4557878

Abstract

We study the relationship between some alternative definitions of the concept of the width of a convex set on the sphere. Those relations allow to characterize whether a convex set on the sphere can pass through a spherical interval by rigid motions. Finally, we give an optimal algorithm to compute the width on the sphere.
Key words: Width, sphere, rigid motions, convex sets.

1 Introduction

In the plane, the width of a finite set of points is the minimum distance between parallel lines of support of the set [8]. This concept and the computation of the width of a finite set of points have applications in several fields such as in robotic (more specifically in collision–avoidance problems [18]), in approximating polygonal curves (see [9], [10] and [11]), etc. Moreover, the width of a set is familiar in Operations Research as a minimax location problem, in which we seek a line (the bisector to the lines of support given the width) whose greatest distance to any point of the set is a minimum.

The definition of the width of a finite set in the plane can be extended to Euclidean spaces of dimension greater than two. So, if we consider a finite set of points P in the space $\mathbf{R}^d$, the width of P is the minimum distance between paralell hyperplanes of support of P [8].

As the width of a finite set in the plane is the width of its convex hull [8], many authors have studied the width of convex polygons, because convex polygons are simple sets and they have many applications in pattern recognition [1], image processing [14] and stock cutting and allocation (see [5], [16] and [6]). By using the rotating caliper technique [15] or geometric transforms [3] it is possible to find the width of a convex polygons in linear time and space.

We will see that it is possible to adapt the rotating caliper technique to design an algorithm for computing the width of a set in the sphere. The study of the width in non-planar surfaces is motived by motion plannig [13], and more concretely a subfield of motion planning of considerable practical interest as it is planning the motion of an articulated robot arm, since, as it is well known, in most cases the points accesible by them are not, in general, in the plane but in a non–planar surface.

G. Strang [17] proved that the width of a convex set in the plane is equivalent to the concept of *door* of the set. The door of a set is the minimun closed interval such that the set can pass through it by a continuous family of rigid motions (translations combined with rotations). Nevertheless, in dimension three this is not true and H. Stark has contructed convex sets which can pass through a door, either square or circular, although no projection of the set will fit in the doorway (see [17]). We will see that, with regard to this problem, the behavior of the sphere is exactly the same as in the plane.

In addition to the problem of the door of a set, there exist another problems that can

be solved knowing the width of a set. For instance, it is easy to see that the *line center* (that line minimizing the maximum distance to each point of a set) is the median (the equidistance parallel to the a pair of parallels) of the pair of support giving the width.

The goal of this paper is to try to generalize these concepts to convex sets on the sphere and, in addition, to seek necessary and sufficient conditions that those convex sets may verify to pass through a spherical interval by rigid motions on this surface, and the relationships between the concepts described above. These conditions and relationships allow us to design an algorithm which solves the problem of the width of a finite set of points on the sphere.

Our generalizations will use the concept of convex set on the sphere. We can define as in [12] that a set C in the sphere is convex if given two points of C the minimun geodesic joining these points is contained in C.

The angular length of a geodesic arc joining the points P and Q on the sphere is the angle between the two radii joining the center of the sphere with the points P and Q respectively. Observe that no convex set on the sphere contains a geodesic arc with angular length greater than $\frac{\pi}{2}$.

2 Width on the sphere

Before trying to generalize the concept of width of a finite set in the plane to the sphere, if we want to give a similar treatment of this idea, we will examine several alternative definitions that are considered in the plane, keeping in mind that we will try to preserve those properties when trying the extension to the sphere. So, firstly, we would replace the idea of lines of support of a set by geodesics of support of a set. In this way, if given a convex set C on the sphere, we will call *meridians of support of* C to the meridians which intersect C and leave the set on one hemisphere. We will call *lune of support of* C to the region delimited by two meridians of support of C that contains C. As, in the sphere, two differents meridians have two points in common and they define only one great circle called equator, thus a lune of support defines one equatorial arc.

According to these definitions, we can say that, given a convex set C on the sphere, the *time width* $\mathcal{H}(C)$ of C is the minimun length between the equatorial arcs defined associated to lunes of support of C.

The main difference between this definition and the definition of width in the plane is that, in the plane, two parallel lines of support have empty intersection, whereas on the sphere two meridians of support have two points in common. If we want to preserve the property that the arcs of support of a convex set have empty intersection, similarly as in the plane, we could give another possible definition. Given a convex set C on the sphere, we will call *parallel of support of* C to a parallel which intersects C and leaves the set on one cap, where a cap is a part of the sphere divided by this parallel. If we use the idea of pair of parallels of support, we will conserve the concept of parallelism that we had in the plane (in the sense that they have empty intersection), but note that parallels in the sphere are not geodesics.

According to the definition above, we can say that given a convex set C on the sphere, the *tropical width* $\mathcal{T}(C)$ of C is the minimun distance between all possible pairs of parallels of support of C. Observe, that with this definition the tropical width of a set in the sphere can be used, as in the plane, in Operations Research as a minimax location problem, in which we seek a great circle (the equator of the parallels of support given the tropical width) whose greatest distance to any point of the set is a minimum.

On the other hand, and following the paper of Strang [17] who proved that the width of a convex set in the plane is the minimum length of an closed interval for the set can pass through it by a continuous family of rigid motions, we can give other definitions of width in the sphere as follows, given a convex set C on the sphere, the *door* $\mathcal{P}(C)$ of C is the minimun length between all possible closed arcs of meridians for the set C can pass through them

by continuous family of rigid motions (translations combined with rotations) on the sphere.

G. Strang proved that the width of a convex set coincides with its door in the plane. But, as it was pointed out in the introduction, in dimension three this is not true and H. Stark has contructed convex sets which can pass through a door, either square or circular, although no projection of the set will fit in the doorway (see [17]. Thus it is interesting to ask if the behavior of the sphere is, in this point, similar to the plane or to the three dimensional space.

In the sphere, we have the following properties

Lemma 1 *Let C be a convex set on the sphere. Then, $\mathcal{P}(C) \leq \mathcal{T}(C)$.*

Proof: It suffices to consider the arc of meridians orthogonal and contained between the parallels which define $\mathcal{T}(C)$. The length of this arc is greater or equal than $\mathcal{P}(C)$ and, obviously, less or equal than $\mathcal{T}(C)$. □

Lemma 2 *Let C be a convex set on the sphere. Then, $\mathcal{T}(C) \leq \mathcal{H}(C)$.*

Proof: Let $\mathcal{H}$ be the lune that defines $\mathcal{H}(C)$. This lune is defined by meridian arcs which intersect C in two points P and Q. We consider the parallels tangent to C in the points P and Q. The distance $\mathcal{T}^*$ between these parallels is equal to $\mathcal{H}(C)$, so $\mathcal{T}(C) \leq \mathcal{T}^* = \mathcal{H}(C)$. □

Therefore, $\mathcal{P}(C) \leq \mathcal{T}(C) \leq \mathcal{H}(C)$. Next theorem says, that, as it happens in the plane, these three numbers agree in the sphere.

Theorem 3 *A convex set C on the sphere can pass through a meridian arc of length $\mathcal{P}(C)$ if and only if $\mathcal{H}(C) \leq \mathcal{P}(C)$.*

Proof: If $\mathcal{H}(C) \leq \mathcal{P}(C)$ and as C is contained in the lune which its equatorial arc has length $\mathcal{H}(C)$, obviously C can pass through this equatorial arc by rigid motions. To prove the converse, assume first that the boundary ∂C of C is smooth, through every boundary point there is a unique tangent line on the sphere, and it varies continuously along ∂C. Let I be an arc of meridian of length $\mathcal{P}(C)$ and denote by S the spherical surface. As C can pass through I, it is possible to define a continuous composition of motions $M : [0,1] \to S$ where $M(0)$ is the situation of C before going into I and $M(1)$ the situation after passing through I. For all $t \in [0,1]$, we can define two applications $f_1 : [0,1] \to [0,\pi]$ and $f_2 : [0,1] \to [0,\pi]$ as follows: $f_1(t)$ and $f_2(t)$ are the angular lengths between the points P_1 and P and the points P_2 and P respectively, where P_1 and P_2 are the intersection of ∂C with the arc I and P is the intersection between the tangents to C in the points P_1 and P_2 (see Figure 1).

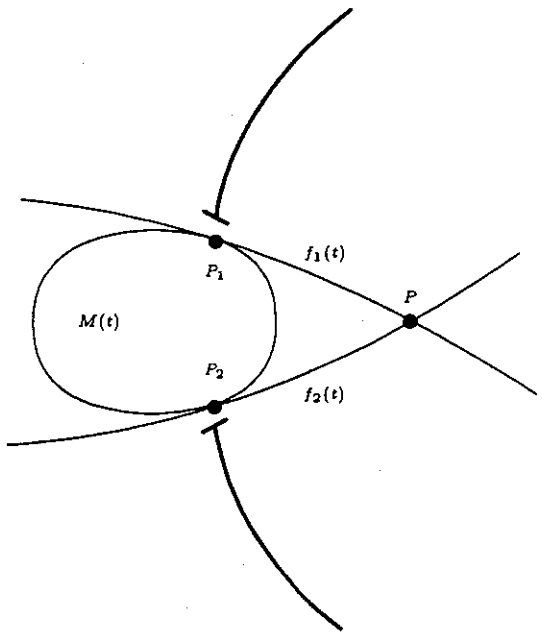


Figure 1

The application $f_1 + f_2 : [0,1] \to [0,2\pi]$ is continuous and $f_1(0) + f_2(0) = 0$ and $f_1(1) + f_2(1) = 2\pi$, so there exists $t^* \in [0,1]$ such that $f_1(t^*) + f_2(t^*) = \pi$.

If $f_1(t^*) = f_2(t^*) = \dfrac{\pi}{2}$, then the meridian arc which defines $\mathcal{H}(C)$ is contained in I, so $\mathcal{H}(C) \leq \mathcal{P}(C)$. Else, $f_1(t^*) - \dfrac{\pi}{2} = \dfrac{\pi}{2} - f_2(t^*)$. Suppose that $f_1(t^*) > \dfrac{\pi}{2}$ and so $f_2(t^*) < \dfrac{\pi}{2}$. Then, the situation is as in Figure 2.

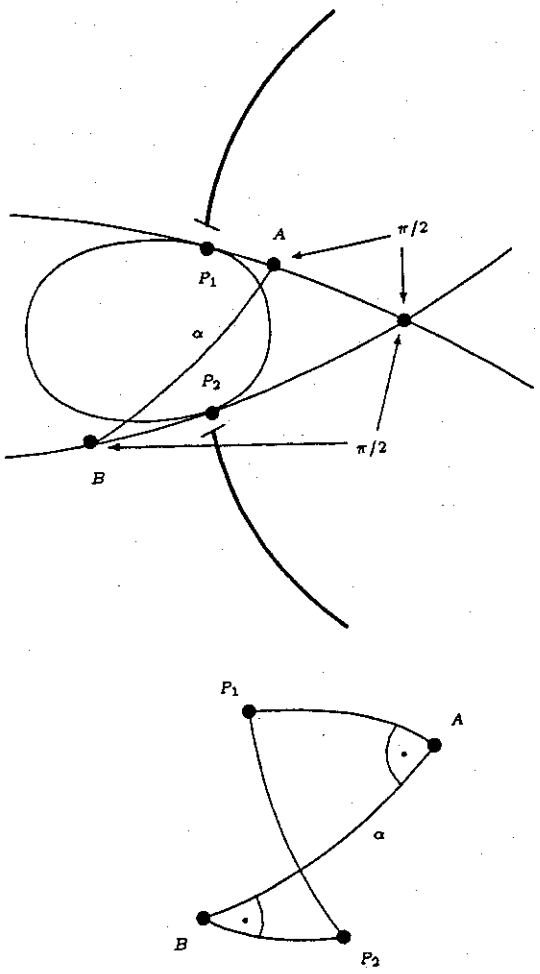


Figure 2

As the angles in the points A and B are of ninety degree, the length of the meridian arc joining P and Q is greater or equal than the length of the meridian arc joining A and B. So, $\mathcal{H}(C) \leq \mathcal{P}(C)$

The conclusion remains true for a convex set C even if ∂C is not smooth. We will proceed introducing a sequence of smooth convex subsets C_n converging to C. As C passes through I so do the C_n and their time widths must satisfy $\mathcal{H}(C_n) < \mathcal{P}(C)$. Therefore, $\mathcal{H}(C) \leq \mathcal{P}(C)$ and the theorem is proved. □

Then, the three definitions of width we have considered agree and we can talk about the width of a set.

As an inmediate consequence of Theorem 3 we get

Corollary 4 *The minimum equatorial arc of a convex set C is included in C.*

3 Algorithm of the width on the sphere

Recall, that in the plane the width of a convex polygon is the minimum distance between parallel lines of support passing through an antipodal vertex-edge pair (to each antipodal vertex-edge pair, we associated the lune define by the meridian containing the edge and that containing the vertex such that the equator arc joins the vertex with the edge). In the sphere this is not true and it can be achieved in an antipodal edge-edge pair as Figure 3 shows, but we have

Lemma 5 *The width of a convex polygon is the minimum distance between meridians of support passing through either an antipodal vertex-edge pair or an edge-edge pair.*

Proof: It is an inmediate consequence of Corollary 4. □

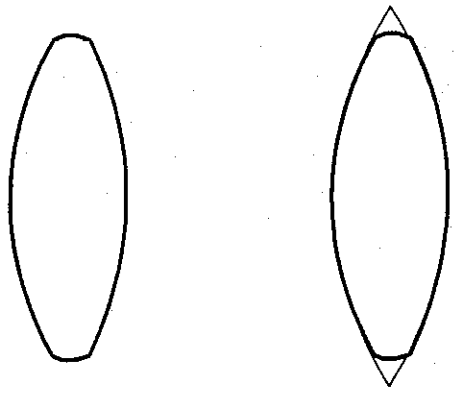


Figure 3

In any case, Lemma 5 says us that it is possible to adapt the rotating caliper algorithm to find the width of a convex polygon C (the number of events is linear). Thus we can give the following algorithm

WIDTH(C)

1.- Find an initial antipodal vertex-edge pair.

2.- If the associated lune to the vertex-edge pair contains C, compute its equator arc, otherwise

compute the equator arc of the pair edge-edge associated to the original vertex-edge pair (this edge-edge pair is defined from the vertex-edge pair by considering the edge incident with the vertex that is not contained in the lune).
3.- Use rotating caliper to generate all pairs as in (1)-(2).
4.- Compute the minimum obtained in previous steps.

It is straightforward to check the following result

Theorem 6 *Algorithm* WIDTH(C) *computes the width of a convex polygon C in optimal linear time.*

References

[1] S. G. AKL AND G. T. TOUSSAINT. *Efficient convex hull algorithms for pattern recognition applications.* Proc. 4th. Int. Joint Conf. on Pattern Recognition (Kyoto, Japan). 1978, pp. 483–487.

[2] J. C. DANA, C. I. GRIMA AND A. MÁRQUEZ. *Convex hull in non–planar surfaces.* 13th European Workshop on Computational Geometry (CG'97). University of Wuerzburg, Germany. 1997.

[3] K. Q. BROWN. *Geometric transform for fast geometric algorithms.* Dep. Comput. Sci., Carnegie–Mellon Univ. 1979.

[4] L. DANZER, G. GRUNBAUM AND V. L. KLEE. *Helly's theorem and its relatives.* Proc. Symp. Pure Math. VII (Providence, R. I.), American Mathematical Society, 1963.

[5] H. FREEMAN. *Computer processing of line–drawing images.* Comput. Surveys (1974), no. 6, pp. 57–97.

[6] H. FREEMAN AND R. SHAPIRA. *Determining the minimum–area encasing rectangle for an arbitrary closed curve.* Comm. ACM. 18 (1975), no. 7, pp. 409–413.

[7] H. HOPF AND W. RINOW. *Über den Begriff der vollständigen differentialgeometrischen Fläche.* Math. Ann. (1931), no. 63, pp. 209–225.

[8] M. E. HOULE AND G. T. TOUSSAINT. *Computing the with of a set.* IEEE Trans. on pattern analysis and machine intelligence, vol. 10, no. 5, 1988, pp. 761–765.

[9] K. ICHIDA AND T. KIYONO. *Segmentation of plane curves.* Trans. Elec. Commun. Eng., Japan, vol. 58–D, 1975, pp. 689–696.

[10] H. IMAI AND M. IRI. *Polygonal approximation of a curve: Formulations and solution algorithms.* Computational Morphology. Amsterdam, The Netherlands: North–Holland, to be published.

[11] Y. KUROZUMI AND W. A. DAVIS. *Polygonal approximation by the minimax method.* Comput. Graphics Image Processing, vol. 19, 1982, pp. 248–264.

[12] K. MENGER. *Urtersuchungen über allgemeine Metrik.* Math. Ann. (1928), no. 100, pp. 75–163.

[13] J. O'ROURKE. *Computational Geometry in C.* Cambridge University Press, 1994.

[14] A. ROSENFELD. *Picture processing by computers.* Academic Press, NY. 1969.

[15] M. I. SHAMOS. *Computational Geometry.* Ph. D. dissertation, Yale Univ. 1978.

[16] J. SKLANSKY. *Measuring concavity on a rectangular mosaic.* IEEE Trans. Comp. C (1972), no. 21, pp. 1355–1364.

[17] G. STRANG. *The width of a chair.* The American Mathematical Monthly, vol. 89, no. 8, 1982 pp. 529–534.

[18] G. T. TOUSSAINT. *Movable separability of sets.* Computational Geometry. Amsterdam, The Netherlands: North–Holland, 1985,pp. 335–375.

Diameter of a set on the cylinder

F.J. Cobos J.C. Dana C.I. Grima A. Márquez

Departamento de Matemática Aplicada I

Universidad de Sevilla (Spain)

e-mail:{cobos|dana|grima|almar}@cica.es

Fax:+34-5-4557878

Abstract

We present an algorithm that computes the diameter of a set of n points in the cylinder in optimal time $O(n \log n)$; this algorithm uses as a fundamental tool the farthest point Voronoi diagram.

1 Introduction

A well-known measure of the spread of a set is its *diameter* (i.e., the maximum distance between two points of the set). Intuitively, a cluster with small diameter has elements that are closely related, while the opposite is true when the diameter is large. This concept has led to several related problems producing a remarkable amount of literature (see, for instance, [1, 6, 7, 17, 18]). But most of the efforts have been concetrated in the plane or euclidean spaces, and, in many cases the set of points in which we are interested are not in an euclidean space but confined to some surface (or a more general space) and the usual techniques are not valid anymore.

It is known that the computation of the diameter of a set of n points in every Euclidean space requires $\Omega(n \log n)$ operations. The usual procedure to compute in optimal time the diameter in the plane uses the fact that the diameter of a set of points is equal to the diameter of its convex hull [9], then it is enough to compute all antipodal pairs and, in a convex polygon, this task can be completed in linear time, thus the total running time of the algorithm is $O(n \log n)$. Unfortunately, this method cannot be used in the space since the number of antipodal pairs in the space is $O(n^2)$. And, in fact it is not known a $O(n \log n)$ algorithm in dimension 3 (as far as we know, the best result for the running time of a deterministic algorithm for the three-dimensional diameter problem is an $(O(n \log^3 n)$ algorithm due to Amato, Goodrich and Ramos [2]). In this paper, we will show that although the procedure followed in the plane cannot be applied in the cylinder, it is possible to get an optimal algorithm in that surface by using the farthest point Voronoi diagram.

The main obstacle in the cylinder is that the convex hull of a set of points is, in general, too big [4], and therefore, it is not usefull as a tool for other problems. In fact, it is not difficult to find examples of sets of points in the cylinder such that their diameters are not equal to the diameters of their convex hulls. Therefore, it is needed another technique to get an optimal algorithm, in this paper our goal is achieved by using the farthest point Voronoi diagram in the cylinder. The structure of this work is as follows, next section will be devoted to summarize some results of [4] about the convex hull of a set of point in the cylinder. In Section 3 we will develop the farthest-

point Voronoi diagram in the cylinder, and in Section 4 we will present our algorithm. We will finish with some conclusions, related problems and open questions.

2 Convex hull in the cylinder

Several extensions of convexity to non-planar surfaces (or to non-euclidean spaces) have been considered in the literature. Most of them are based on metrical concepts, and more concretely, in the family of geodesics of the surface. In order to describe the family of geodesics, as usual, we identify the cylinder with the quotient space obtained from the plane by identifying those points with the same ordinate such that their abscisae differ in an integer number. With the metric obtained from this definition, the geodesics joining two points in the cylinder can be identified with the segments in the plane joining a fixed representative of one of the points and all of the representatives of the other point. We say that generatrices $\{(x,y) : x = x_0\}$ and $\{(x,y) : x = x_0 + 1/2\}$ are *opposite generatrices.*

Using this representation, we can define the *strip* of a set of points $P = \{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$ in the cylinder with $y_1 \leq y_2 \leq \ldots \leq y_n$ as the open strip O delimited by the maximal circles in the extreme points of P with respect to the ordinates ($O = \{(x,y) : y_1 < y < y_n\}$). Equally, we define the *m-top* of P as the minimal arc containing all points of P with the ordinate y_n if that arc is shorter than a half of the circle or the whole maximal circle if that arc is greater than a half of the circle or the single point (x_n,y_n) otherwise (equivalently the *m-bottom*).

Then, as Hopf-Rinow's Theorem [10] proves that there exists always the shortest geodesic joining two points, we can define as in [14] that $C \subseteq S$ is *metrically convex* if given two points of C the minimum geodesic in S joining those points is contained in C. And, as usual, given a set P of points in S, the *metrically convex hull* of P is the smallest metrically convex set containing P.

It is possible to give the following characterization of the metrically convex hull

Theorem 1 [4]. *The metrically convex hull of a set of N points P in the cylinder is*

1. *The convex hull of P in the plane if P is contained between two oppossite generatrices.*
2. *The open strip delimited by the points P union the m-top and the m-bottom of P otherwise.*

Moreover, this metrically convex hull can be computed in $O(N \log N)$ time in the first case and in linear time in the second case, and it can be decided in which one of the cases we are in linear time.

Thus, Theorem 1 says that in many cases convex hull is too big for many purposes. In fact Figure 1 shows a set of points in the cylinder such that the diameter of the set is not equal to the diameter of its convex hull.

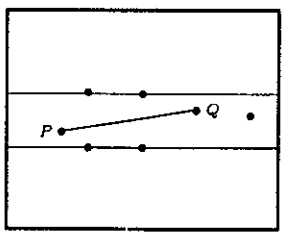


Figure 1

3 Voronoi diagrams on the cylinder

As it has been said in the introduction, the main tool in our algorithm to compute the

diameter will be the farthest point Voronoi diagram. As in the euclidean spaces, given a set of points S in the cylinder, we denote by $V_f(i)$ the locus of points farther to $x_i \in S$ than to any other point of S. The set of all those loci is called the *farthest point Voronoi diagram* of S, $vor_f(S)$. Several methods to compute that structure are known in the plane and there exists a direct method, based on the divide and conquer scheme analogous to the algorithm for the closest-point diagram, which achives the result in optimal $O(n \log n)$ time. On the other hand, Mazón presented in [13] an optimal algorithm to compute the closest-point Voronoi diagram of a set of points in the cylinder. Her method to considers three copies of the cylinder,and it constructs the diagram of the sets of $3n$ points, the diagram in the cylinder is the resulting diagram in the central copy. Therefore, it is not difficult to see that this method cannot be used to generate the farthest-point Voronoi diagram in the cylinder. Then we will try the divide and conquer approach.

Lemma 2 *The bisector of the points $P = (x_1, y_1)$, $Q = (x_2, y_2)$ in the cylinder with $x_1 < x_2$ with $y_1 < y_2$, is given by the bisectors in the plane of the points P and Q and Q and $P' = (x_1 + 1, y_1)$ (see Figure 2).*

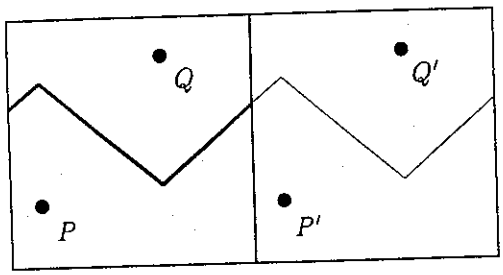


Figure 2

As far as the key step in the divide and conquer algorithm is to construct the dividing chain, we give some properties of that chain. In this order, we suppose that the original set S has been split in two parts S_1 and S_2 by a parallel c and if $x_i \in S_j$ $j = 1, 2$, we denote by $V_f^j(x_i)$ to its region in $vor_f(S_j)$. Then,

Lemma 3 *If $V_f^1(x_i) \cap V_f^2(x_j) \cap c \neq \emptyset$, then the bisector between x_i and x_j appears in the dividing chain of S_1 and S_2.*

Lemma 4 *The orthogonal projection of the dividing chain of S_1 y S_2 on c is a homeomorphism*

Lemma 3 is the key to construct an algorithm in the cylinder similar to the algorithm in the plane.
Algorithm DIVID-CHAIN(S_1, S_2, c):
(1) For $p \in c$. Find $x_1 \in S_1$ y $x_2 \in S_2$ such that $p \in V_f^1(x_1) \cap V_f^2(x_2) \cap c$. Let $x = x_1$, $y = x_2$.
(2) Construct the bisector between x and y.
(3) Determine the portion of bisector computed in (2) that is in $V_f^1(x) \cap V_f^2(y)$.
(4) Compute the extremes of the portion already computed of the dividing chain and update the points x and y.

Lemma 5
Algorithm DIVID-CHAIN(S_1, S_2, c) *computes the dividing chain between S_1 and S_2, subsets of S linearly separated by parallel c in linear time.*

Then we conclude

Theorem 6 *The farthest-point Voronoi diagram of n points in the plane can be constructed in optimal $O(n \log n)$ time.*

4 Diameter of a sets of points in the cylinder

Obviously, if the diameter of S is $d(u, v)$ for certain $u, v \in S$ then $u \in V_f(v)$. Therefore,

the algorithm to compute the diameter will be

Algorithm DIAMETER(S)
(1) Construct the farthest-point Voronoi diagram of S.
(2) Localize in which region of the diagram is each point of S.
(3) Compute the distance between each point of S and the point defining the region obtained in (2).
(4) Report the maximum obtained in (3) as the diameter.

It is straightforward to check the validity of the algorithm DIAMETER(S) and then we have

Theorem 7 *It is possible to compute the diameter of a set of n points in the cylinder in optimal $O(n \log n)$.*

5 Open questions

Although an optimal algorithm to compute the diameter is presented, some open question arise related to the problem considered in this work.

First of all, it would be interested to find a structure that, as the convex hull in the plane, allows to find from it the diameter in the cylinder in linear time (observe that from the farthest-point Voronoi diagram we find the diameter in $O(n \log n)$ time). Moreover, it seems to be that our tecnique can be applied to other surfaces but building the farthest-point Voronoi diagram colud be a difficult task.

Another interesting question that has been studied extensively in Euclidean spaces is, how many times can the maximum distance between n points occur? It is known that in the plane it can occur at most n times [5], and in the space $2n-2$ times [8]. The case of the cylinder is not the same as in the plane since it is possible to give a structure where the maximum can occur $4/3n$. This structure split the n points in three subsets of $n/3$ points each and each of those subsets are a regular polygon in a parallel in such a way that those polygons in the top and in the bottom parallels have their vertices on the same meridians and the other polygon has its vertices on the equidistan meridians to those considered before, see Figure 3.

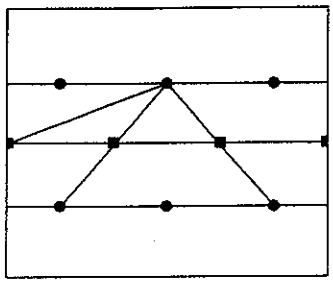


Figure 3

It remains to solve if this example is optimal or to find better bounds.

On the other hand, it is possible to answer completely a related question how many times can the maximum distance between n points occur?

Theorem 8 *The minimum distance between n points in the plane can occur at most $3n-6$ times.*

Proof: It is easy to see that the graph of the minimum distance between points in the cylinder is planar. Thus, by Euler's formula $3n-6$ is an upperbound and Figure 4 shows that this upperbound can be achieved. □

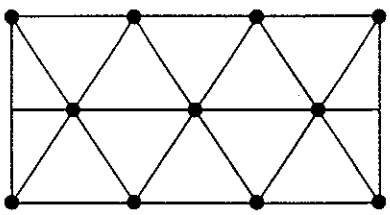


Figure 4

References

[1] S.G. AKL AND G.T. TOUSSAINT. *Efficient convex hull algorithms for*

pattern recognition applications. Proc. 4th. Int. Joint Conf. on Pattern Recognition (Kyoto, Japan). 1978, pp. 483–487.

[2] N.M. AMATO, M.T. GOODRICH AND E.A. RAMOS. *Parallel algorithms for higher-dimensional convex hulls*. Proc. 35th. Annual IEEE Sympos. Found. Comput. Sci. 1994, pp. 683–694.

[3] F.J. COBOS, J.C. DANA, C.I. GRIMA Y A. MÁRQUEZ. *Voronoi diagrams in non–planar surfaces* . Preprint.

[4] J.C. DANA, C.I. GRIMA Y A. MÁRQUEZ. *Convex hull in non–planar surfaces*. 13th European Workshop on Computational Geometry (CG'97). University of Wuerzburg, Germany. 1997.

[5] P. ERDÖS. *On sets of distances of n points*. Amer. Math. Monthly (1946), no. 53, pp. 248–250.

[6] H. FREEMAN. *Computer processing of line–drawing images*. Comput. Surveys (1974), no. 6, pp. 57–97.

[7] H. FREEMAN Y R. SHAPIRA. *Determining the minimum–area encasing rectangle for an arbitrary closed curve*. Comm. ACM. **18** (1975), no. 7, pp. 409–413.

[8] B. GRÜNBAUM. *A proof of Vazsonyi's conjecture*. Bull. Res. Council Israel (1956), no. 6(A), pp. 77–78.

[9] J.G. HOCKING Y G.S. YOUNG. *Topology*. Addison–Wesley, Reading, MA, 1961.

[10] H. HOPF AND W. RINOW. *Über den Begriff der vollständigen differential-geometrischen Fläche*, Comentarii Math. Helvetici (1931), no. 3, 209–225.

[11] M.E. HOULE Y G.T. TOUSSAINT. *Computing the width of a set*. IEEE Trans. on pattern analysis and machine intelligence, vol. 10, no. 5, 1988, pp. 761–765.

[12] D.T. LEE. *Farthest neighbor Voronoi diagrams and applications*, Tech. Rep. No. 80–11–FC–04, Dpt. EE/CS, Northwestern Univ., 1980b.

[13] M.L. MAZÓN. *Diagramas de Voronoi en caleidoscopios*. Ph.D. Thesis, Dpto. de Matemáticas, Estadística y Computación. Univ. de Cantabria, Santander 1992.

[14] K. MENGER. *Urtersuchungen über allgemeine Metrik*, Math. Ann. (1928), no. 100, 75–163.

[15] J. O'ROURKE. *Computational Geometry in C*. Cambridge University Press, 1994.

[16] F.P. PREPARATA Y M.I. SHAMOS. *Computational Geometry. An Introduction*. Springer-Verlag, 1985.

[17] A. ROSENFELD. *Picture processing by computers*. Academic Press, NY. 1969.

[18] J. SKLANSKY. *Measuring concavity on a rectangular mosaic*. IEEE Trans. Comp. C (1972), no. 21, pp. 1355–1364.

Testing Roundness of a Polytope and Related Problems

(extended abstract)

Artur Fuhrmann *

Abstract

In this paper we study the problem of computing the smallest-width annulus for a convex polytope in $\mathbb{R}^d$. This generalizes a topic from tolerancing metrology to higher dimensions, of which the planar case has been investigated by SWANSON, LEE and WU. We show this problem to be equivalent to the problem of computing different variants of Hausdorff-minimal spheres. We can formulate the latter problem as a LP-type problem with combinatorial dimension $2(d+1)$ (for the planar case: 4) and prove the uniqueness of Hausdorff-minimal spheres for arbitrary convex bodies.

The result is an expected linear-time algorithm in the size of the input, for all these problems. In conclusion we describe positive and negative results concerning an extension from spheres to ellipsoids.

1 Introduction and Definitions

In manufactoring, people are interested in ascertaining the deviation of a produced physical shape from its technical specification. This deviation is then compared with a given tolerance. The introduction of new quality-testing machines into the production process let arise an interest in algorithms that compute certain *characteristic values* of physical shapes. One of these, recommended by the American National Standarts Institute (ANSI), is the so-called *Minimum Radial Separation (MRS)* that is determined by the minimal possible difference of radii of two concentric spheres containing the surface of the object in between. LE and LEE[1] considered a planar convex polygon as an input and presented an $O(n^2)$-time algorithm. They also conjectured $O(n \log n)$ to be a lower bound for the time-complexity of the problem. SWANSON, LEE and WU[6] improved this result by an factor of n and presented a linear-time algorithm. Both works use Voronoi diagrams, a fact that complicates a generalization to higher dimensions.

Another type of problems considered here are problems concerning approximations of convex polytopes by spheres. They arise as subproblems in many fields, such as collision detection, motion-planning or ray-tracing. Our solution can be used in different ways. For example, we are able to approximate convex polyhedral patches by spherical caps. The planar case of this problem arises in the build-up phase of the *Arc-tree*, a spatial data-structure that is described in the book of SAMET[3].

We start by introducing some definitions and notations. A sphere in d-space with radius r and center $\mathbf{c}$ is given by $\mathcal{S}_r(\mathbf{c}) := \{\mathbf{x} \in \mathbb{R}^d \mid (\mathbf{x}-\mathbf{c})^T(\mathbf{x}-\mathbf{c}) - r^2 \leq 0\}$. As mentioned before, we use the Hausdorff-metric $\delta^H(.\,,.)$ to measure the quality of an approximation. The exact definition and some basic properties follow later. This paper addresses the following problems:

Problem 1 (minimum-width annulus).
Let $\mathcal{P} \subset \mathbb{R}^d$ be a convex polytope.
Find an optimal pair $(\mathcal{S}, \mathcal{S}')$ of concentric spheres, with $\mathcal{S} \subseteq \mathcal{P} \subseteq \mathcal{S}'$, that minimizes the difference of their radii.

Problem 2 (Hausdorff-minimal spheres).
Let $\mathcal{P} \subset \mathbb{R}^d$ be a convex polytope.
Find an optimal sphere $\mathcal{S}^*$ that minimizes $\delta^H(\mathcal{S}^*, \mathcal{P})$, subject to

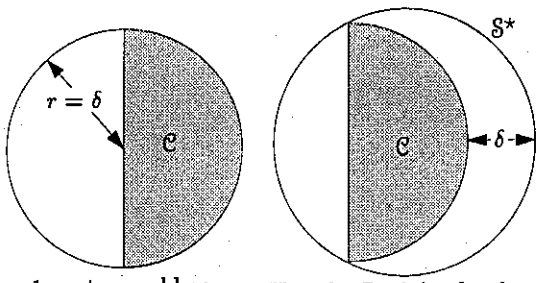
(i) no further constraints.

(ii) the constraint '$\mathcal{S}^* \subseteq \mathcal{P}$'.

(iii) the constraint '$\mathcal{S}^* \supseteq \mathcal{P}$'.

Both problems intend not only to find the specific geometric figure, but also the optimal value ε^* of the difference of radii or δ^H, respectively. For a better understanding consider problem 2.*(iii)*. It could easily be mixed up with the well-known *1-center* problem that asks for a sphere with minimal radius, containing a set of points. There are convex bodies, for which we can get a good improvement to the Hausdorff-distance by only slightly increasing the radius of the smallest enclosing sphere.

*Universität des Saarlandes, Fachbereich 14, Informatik, Lehrstuhl Prof. G. Hotz, Postfach 151150, 66041 Saarbrücken, Germany. Email: fuhrmann@cs.uni-sb.de, Fax: +49 681 302 4421.



1-center problem Hausdorff-minimal sphere

Subsequently, we review some well-known facts from convex geometry that can also be found in standard textbooks, as BONNESEN, FENCHEL[7]. Let $\|.\|$ be any norm of $\mathbb{R}^d$. The *Hausdorff-distance* of two bounded sets of points $S, S' \subset \mathbb{R}^d$ is given by

$$\delta^H(S,S') := \max\{\overrightarrow{\delta^H}(S,S'), \overrightarrow{\delta^H}(S',S)\},$$

with $\overrightarrow{\delta^H}(S,S') := \sup_{\mathbf{x}\in S} \inf_{\mathbf{y}\in S'} \| \mathbf{x}-\mathbf{y} \|$ which is called the *directed Hausdorff-distance*. The Hausdorff-distance between two convex bodies $\mathcal{C}, \mathcal{C}' \subset \mathbb{R}^d$ is easily explained by using the notion of the *parallel-body of* $\mathcal{C}$ *at distance* ε:

$$\mathcal{C}_\varepsilon := \{\mathbf{x} \in \mathbb{R}^d \mid \exists \mathbf{y} \in \mathcal{C} : \|\mathbf{x}-\mathbf{y}\| \le \varepsilon\}$$

Note, that $\overrightarrow{\delta^H}(\mathcal{C},\mathcal{C}') \le \varepsilon$ is equivalent to $\mathcal{C} \subseteq \mathcal{C}'_\varepsilon$. The analogous statement with equality is true, iff there exists no such inclusion for any parallel-body at a smaller distance than ε.

A hyperplane h is called *supporting for* $\mathcal{C}$, if $h \cap \mathcal{C} \neq \emptyset$ holds and $\mathcal{C}$ completely lies in a closed halfspace defined by h. Convex bodies have the property, that there is a supporting hyperplane in every point of their surface. MINKOWSKI used this property to define an arbitrary convex body $\mathcal{C}$ by a *supporting function* $s_\mathcal{C}$, given by

$$s_\mathcal{C}(\mathbf{n}) := \sup_{\mathbf{x}\in\mathcal{C}} \mathbf{n}^T\mathbf{x}, \qquad \text{for all } \mathbf{n} \in \mathbb{R}^d.$$

For example, a sphere $\mathcal{S}_r(\mathbf{c})$ can be defined by $s(\mathbf{n}) := \mathbf{c}^T\mathbf{n} + r\sqrt{\mathbf{n}^T\mathbf{n}}$. The following properties will be useful in section 3.

Observation 1. *Let* $\mathcal{C}, \mathcal{C}' \subset \mathbb{R}^d$ *be two convex bodies and* $s_\mathcal{C}, s_{\mathcal{C}'}$ *their corresponding supporting functions. The following statements hold:*

(i) $\mathcal{C} \subseteq \mathcal{C}' \iff s_\mathcal{C}(\mathbf{n}) \le s_{\mathcal{C}'}(\mathbf{n})$ *for all* $\mathbf{n} \in \mathbb{R}^d$.

(ii) *The supporting function of the parallel-body of* $\mathcal{C}$ *at distance* ε *is given by* $s_{\mathcal{C}_\varepsilon}(\mathbf{n}) := s_\mathcal{C}(\mathbf{n}) + \varepsilon\|\mathbf{n}\|$.

2 LP-type Problems

In this section we will give the formal definition and review the most important facts about *LP-type* problems, first introduced by SHARIR and WELZL[5].

Definition 1. Let an optimization problem be specified by a pair (C, w) consisting of a set C of constraints and a function $w : 2^C \longmapsto \mathcal{W}$ with values in a linearly ordered set $(\mathcal{W} \cup \{-\infty\}, \le)$, with $-\infty < w$ for all $w \in \mathcal{W}$. A minimization problem (C, w) is *LP-type*, if the following constraints are satisfied:

Monotonicity. For any A, B with $A \subseteq B \subseteq C$, we have $w(A) \le w(B)$.

Locality. For any $A \subseteq B \subseteq C$ with $w(A) = w(B) \neq -\infty$ and any $c \in C$, the following implication is true:
$w(B) < w(B \cup \{c\}) \implies w(A) < w(A \cup \{c\})$.

We set $w(B) := -\infty$, if we want the value for a set $B \subseteq C$ to be *undefined*. A subset A is called a *basis*, if $w(A') < w(A)$ holds for all proper subsets $A' \subset A$. A basis $A \subseteq B$ is called a *basis for* B, if it has the property $w(A) = w(B)$. The *combinatorial dimension* of (C, w) is the maximal cardinality of any basis and is denoted by $\dim(C, w)$. A LP-type problem is called *basis-regular*, if all bases have the same cardinality.

We need some further notions. A constraint $c \in C$ is called *violated* by $B \subseteq C$, if $w(B) < w(B \cup \{c\})$ is true. If $w(B) > w(B \setminus \{c\})$ holds, c is said to be *extreme* in B. Every basis for B contains all extreme constraints of B.

The intention behind LP-type problems is the existence of problems that cannot be formulated in the form of a linear program, but have very similar combinatorial properties. A classical example is the 1-center problem. (Compare the work of SEIDEL[4] with WELZL[8].)

The time complexity for solving a LP-type problem (C, w) is expected linear to $|C|$, if the following two primitive operations take constant time (in size of $|C|$). Better bounds, mainly a subexponential one for the dependence on the combinatorial dimension, are known for problems that satisfy basis-regularity.

Violation test. Decide whether a constraint c is violated by a basis B, i.e. whether $w(B) < w(B \cup \{c\})$ holds.

Basis computation. Compute a basis for a constraint c and a basis B.

The properties of the above framework lead to the fact, that a typical algorithm that solves a LP-type

problem works in a dual manner. The posed minimization problem is solved by steadily increasing an intermediate result.

3 Equivalence

In this section we want to show, that problem 1 and all three variants of problem 2 can be solved at once, by finding a solution for one of these four problems. This is correct, only if the Euclidean norm underlies the Hausdorff-distance. We assume the latter for the rest of this paper. The following result states an interesting geometric correspondence between Hausdorff-minimal spheres and minimum-width annuli for arbitrary convex bodies.

Theorem 1. *Let $\mathcal{C} \subset \mathbb{R}^d$ be a convex body, $\varepsilon \in \mathbb{R}$ and let all subsequent spheres have the same center $\mathbf{c} \in \mathcal{C}$. The following statements are equivalent, if the Euclidean norm underlies the metric $\delta^H(.\,,.)$.*

(i) *There exists a sphere $\mathcal{S}_{r+\varepsilon} \supseteq \mathcal{C}$ with $\delta^H(\mathcal{S}_{r+\varepsilon}, \mathcal{C}) \leq \varepsilon$.*

(ii) *There exists a sphere $\mathcal{S}_r \subseteq \mathcal{C}$ with $\delta^H(\mathcal{S}_r, \mathcal{C}) \leq \varepsilon$.*

(iii) *There exists a sphere $\mathcal{S}_{r+\varepsilon/2}$ with $\delta^H(\mathcal{S}_{r+\varepsilon/2}, \mathcal{C}) \leq \varepsilon/2$.*

(iv) *There exists a pair $(\mathcal{S}_r, \mathcal{S}_{r+\varepsilon})$ of concentric spheres with $\mathcal{S}_r \subseteq \mathcal{C} \subseteq \mathcal{S}_{r+\varepsilon}$.*

Proof. First, we prove the following equivalence:

$$\mathcal{S}_{r+\varepsilon} \subseteq \mathcal{C}_\varepsilon \;\Leftrightarrow\; \mathcal{S}_r \subseteq \mathcal{C}, \qquad \text{for all } r \geq \varepsilon \geq 0. \quad (1)$$

This can easily be shown by using supporting functions and their properties as introduced in section 1. Let $\mathcal{S} := \mathcal{S}_{r+\varepsilon}$, then we have for all $\mathbf{n} \in \mathbb{R}^d$:

$$\begin{aligned} s_{\mathcal{C}_\varepsilon}(\mathbf{n}) = s_{\mathcal{C}}(\mathbf{n}) + \varepsilon\sqrt{\mathbf{n}^T\mathbf{n}} \geq s_{\mathcal{S}}(\mathbf{n}) \\ \Leftrightarrow \qquad s_{\mathcal{C}}(\mathbf{n}) \geq s_{\mathcal{S}}(\mathbf{n}) - \varepsilon\sqrt{\mathbf{n}^T\mathbf{n}} \quad (2) \\ = \mathbf{c}^T\mathbf{n} + r\sqrt{\mathbf{n}^T\mathbf{n}} \end{aligned}$$

(i) $\Rightarrow$ (ii). Suppose statement *(i)* is true, i.e. there exists a sphere $\mathcal{S}_{r+\varepsilon}$ with $\mathcal{C} \subseteq \mathcal{S}_{r+\varepsilon} \subseteq \mathcal{C}_\varepsilon$. Since we have fact (1), this is equivalent to $\mathcal{S}_r \subseteq \mathcal{C} \subseteq \mathcal{S}_{r+\varepsilon}$ which implies claim *(ii)*.

(ii) $\Rightarrow$ (iii). We need to prove $\mathcal{C} \subseteq \mathcal{S}_{r+\varepsilon}$ and $\mathcal{S}_{r+\varepsilon/2} \subseteq \mathcal{C}_{\varepsilon/2}$. The first part follows directly from $\delta^H(\mathcal{S}_r, \mathcal{C}) \leq \varepsilon$. By considering formula (1) with factor $\varepsilon/2$ we see, that the second part is equivalent to $\mathcal{S}_r \subseteq \mathcal{C}$.

(iii) $\Rightarrow$ (iv). This corresponds with the previous thought.

(iv) $\Rightarrow$ (i). Here we have $\mathcal{S}_r \subseteq \mathcal{C} \subseteq \mathcal{S}_{r+\varepsilon}$ which implies $\mathcal{S}_{r+\varepsilon} \subseteq \mathcal{C}_\varepsilon$, after an application of (1). This satisfies statement *(i)*. □

Remark 1. As an immediate consequence of the theorem, all four statements remain equivalent, if '$\leq$' is changed to '='. This settles the equivalence between all four problems, as pointed out in the beginning of the section.

If we take a close look at formula (2), we observe a correspondence between the geometric object and the norm underlying the Hausdorff-distance. This means, that theorem 1 remains correct, if we change the norm to L_1 and the sphere to the L_1-unit-sphere, for example.

4 Formulation as a LP-type problem

4.1 Formulation

In this section we want to show how to formulate problem 2.*(iii)* as a LP-type problem. This settles the time complexity of problem 1 and 2 to be linear in the size of the complexity of description of the given polytope.

The input, a polytope $\mathcal{P} \subset \mathbb{R}^d$, is expected to be given in two ways: As the set $P := \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ of its vertices and as the set $H := \{h_1, \ldots, h_m\}$ of its defining hyperplanes. All single objects differ from each other. For each h_i we have the corresponding normal vector $\mathbf{n}_i$ of unit length, pointing out of $\mathcal{P}$. We introduce the following parameterization for H denoted by $H(\varepsilon) := \{h_1(\varepsilon), \ldots, h_m(\varepsilon)\}$:

$$h_i(\varepsilon) := \{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{n}_i^T\mathbf{x} \leq k_i + \varepsilon\},$$

with $H^-(0) := \bigcap_{i=1}^m h_i^-(0) = \mathcal{P}$.

Observation 2. *Let $\mathcal{S} \supset \mathcal{P}$ be a sphere and $\varepsilon > 0$. It holds:*

$$\delta^H(\mathcal{S}, \mathcal{P}) \leq \varepsilon \quad \Longleftrightarrow \quad P \subset \mathcal{S} \subset H^-(\varepsilon).$$

The crucial point is the existence of cylindrical and spherical parts on the surface of $\mathcal{P}_\varepsilon$. They have a smaller radius than $\mathcal{S}$, hence they cannot intersect with $\mathcal{S}$ and can be ignored.

We now define the pair (C, w) as follows: The set of constraints is $C := P \cup H$. For a set $B \subseteq C$ we denote $B_P := B \cap P$ and $B_H := B \cap H$. We want to minimize a factor ε, such that there exists a sphere $\mathcal{S}$ with $B_P \subset \mathcal{S} \subset B_H^-(\varepsilon)$. The optimal factor $\varepsilon^\star$ for B is denoted by $\varepsilon^\star(B)$. If there exist several optimal spheres, we define the smallest one to be the optimal sphere $\mathcal{S}^\star(B)$, with radius $r^\star(B)$. That means, that we lexicographically minimize the pair (ε, r) and denote the optimal pair for B by $(\varepsilon^\star, r^\star)(B)$.

In certain cases, the optimal sphere may not exist and degenerate to a hyperplane. In order to avoid

this, we restrict the maximal possible radius of any sphere by a very large constant R. This does not affect the over-all solution of the problem. Formally, we set $\mathcal{W} := \mathbb{R} \times \mathbb{R}$, and '$\leq$' to the standard lexicographic order. The function w is given as follows:

$$w(B) := \begin{cases} -\infty & \text{if } B_P = \emptyset \text{ or } B_H = \emptyset, \\ (\varepsilon^\star, r^\star)(B) & \text{else.} \end{cases}$$

4.2 Uniqueness

We are left to show, that the minimization problem (C, w), defined as above, satisfies the conditions of definition 1 and has restricted combinatorial dimension. The monotonicity of (C, w) is easy to see and requires no further explanation. For locality we discover, that $w(A) = w(B) > -\infty$, for $A \subseteq B \subseteq C$, implies $\mathcal{S}^\star(A) = \mathcal{S}^\star(B)$, if $\mathcal{S}^\star(A)$ and $\mathcal{S}^\star(B)$ are unique.

In order to show uniqueness, we use an old tool from convex geometry that was already used by POST[2] and WELZL[8]. See also BONNESEN, FENCHEL[7].

Definition 2. Let $\mathcal{S}_0, \mathcal{S}_1 \subset \mathbb{R}^d$ be two spheres given implicitly by $f_i(\mathbf{x}) := (\mathbf{x} - \mathbf{c}_i)^T(\mathbf{x} - \mathbf{c}_i) - r_i^2 \leq 0$, for $i = 0, 1$. The *convex combination of* $\mathcal{S}_0$ *and* $\mathcal{S}_1$ *with factor* $\lambda \in \mathbb{R}$ is defined by:

$$\begin{aligned} &\mathrm{comb}_\lambda(\mathcal{S}_0, \mathcal{S}_1) := \\ &\{\mathbf{x} \in \mathbb{R}^d \mid f_\lambda(\mathbf{x}) := \lambda f_0(\mathbf{x}) + (1-\lambda) f_1(\mathbf{x}) \leq 0\}. \end{aligned}$$

It can be checked, that $\mathrm{comb}_\lambda(\mathcal{S}_0, \mathcal{S}_1)$ is really a sphere for $\lambda \in [0; 1]$, iff $\mathcal{S}_0 \cap \mathcal{S}_1 \neq \emptyset$. We need the following characterizing lemma.

Lemma 1. *Let* $\mathcal{S}_0 \neq \mathcal{S}_1$ *and* $\mathcal{S}_\lambda := \mathrm{comb}_\lambda(\mathcal{S}_0, \mathcal{S}_1)$ *be as above, and let their radii be denoted by* r_0, r_1 *and* r_λ, *respectively. If* $\mathcal{S}_0 \cap \mathcal{S}_1 \neq \emptyset$, *we have for* $0 < \lambda < 1$:

(i) $\mathcal{S}_0 \cap \mathcal{S}_1 \subseteq \mathcal{S}_\lambda \subseteq \mathcal{S}_0 \cup \mathcal{S}_1$.

(ii) $\partial\mathcal{S}_\lambda \cap (\partial\mathcal{S}_0 \cup \partial\mathcal{S}_1) = \partial\mathcal{S}_0 \cap \partial\mathcal{S}_1$.

(iii) $r_\lambda < \max\{r_0, r_1\}$.

Proof. (i). Let $\mathbf{x} \in \mathcal{S}_0 \cap \mathcal{S}_1$, i.e. $f_i(\mathbf{x}) \leq 0$ for $i = 0, 1$. Because of the positiveness of λ and $(1-\lambda)$, the immediate result is $f_\lambda(\mathbf{x}) \leq 0$, as well as $\mathbf{x} \in \mathcal{S}_\lambda$. The assumption $\mathbf{x} \notin \mathcal{S}_0 \cup \mathcal{S}_1$, i.e. $f_i(\mathbf{x}) > 0$, directly implies $f_\lambda(\mathbf{x}) > 0$.

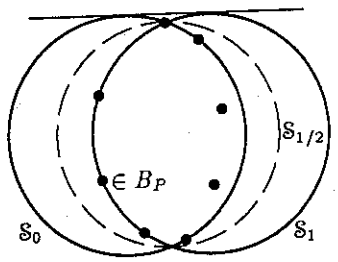
(ii). Let $\mathbf{x} \in \partial\mathcal{S}_0 \cap \partial\mathcal{S}_1$ which is equivalent to $f_i(\mathbf{x}) = 0$, for $i = 0, 1$. This means, that $f_\lambda(\mathbf{x}) = 0$ and $\mathbf{x} \in \partial\mathcal{S}_\lambda \cap (\partial\mathcal{S}_0 \cup \partial\mathcal{S}_1)$. No further points of $\partial\mathcal{S}_0 \cup \partial\mathcal{S}_1$ are allowed to lie on the surface of $\mathcal{S}_\lambda$. Consider such a point $\mathbf{x}$, with $f_i(\mathbf{x}) \neq 0$ for $i = 0$ or 1. By the same arguments as before, we have $f_\lambda(\mathbf{x}) \neq 0$.

(iii). Can be checked algebraically. Details omitted. □

Theorem 2. *Let* (C, w) *and* $\mathcal{S}^\star(B)$ *be defined as above, with* $B \subseteq C$ *and* $w(B) > -\infty$. *Then the sphere* $\mathcal{S}^\star(B)$ *is unique.*

Proof. Let $\varepsilon^\star := \varepsilon^\star(B)$. We suppose, that there exist two different spheres $\mathcal{S}_0, \mathcal{S}_1$ that are optimal for B. That is, they have the same radius $r^\star(B)$ and the property $B_P \subset \mathcal{S}_0, \mathcal{S}_1 \subset B_H^-(\varepsilon^\star)$ holds.

Since both spheres are distinct, they have to intersect in a way, that none of both is equal to $\mathrm{conv}(\mathcal{S}_0, \mathcal{S}_1)$. We consider $\mathcal{S}_{1/2} := \mathrm{comb}_{1/2}(\mathcal{S}_0, \mathcal{S}_1)$ which contains B_P and lies inside $\mathcal{S}_0 \cup \mathcal{S}_1$, as stated in lemma 1.*(i)*. Moreover, lemma 1.*(iii)* implies, that $\mathcal{S}_{1/2}$ has a radius $< R$ and therefore is feasible in our context. We use the observation, that no common point of $\mathcal{S}_0$ and $\mathcal{S}_1$ can lie on the surface of their convex hull. Then lemma 1.*(ii)* implies, that $\mathcal{S}_{1/2}$ lies in the interior of $\mathrm{conv}(\mathcal{S}_0, \mathcal{S}_1)$ and does not touch its surface. Hence, no hyperplane of $B_H(\varepsilon^\star)$ can support $\mathcal{S}_{1/2}$. This contradicts optimality, since $\varepsilon^\star$ can be reduced. □



Remark 2. Actually, we need the lexicographical minimization of the radius only for sets B with $\varepsilon^\star(B) \leq 0$, but such values cannot appear as a Hausdorff-distance (unless both objects are equal). By using similar techniques and the general notions for arbitrary convex bodies, as defined in section 1, we can also prove the following general result.

Theorem 3. *Let* $\mathcal{C} \subset \mathbb{R}^d$ *be a convex body. The optimal sphere* $\mathcal{S}^\star$ *that minimizes* $\varepsilon^\star := \delta^H(\mathcal{S}^\star, \mathcal{C})$, *subject to* $\mathcal{S}^\star \supseteq \mathcal{C}$, *is unique.*

This holds for any norm underlying the Hausdorff-distance. In the special case of the Euclidean norm and with the help of theorem 1, we showed the uniqueness of all three kinds of Hausdorff-minimal spheres and of the minimum-width annulus for arbitrary convex bodies.

4.3 Combinatorial dimension

The last question not answered concerns the size of the combinatorial dimension of (C, w). We use the fact, that a constraint c, that is extreme for $B \subseteq C$, is also geometrically in a kind of extreme position. We claim:

Lemma 2. *Let (C, w) be defined as above, and let c be an extreme constraint for a subset $B \subseteq C$ with $w(B) > -\infty$. Then the geometric object corresponding to c touches the surface of $\mathcal{S}^*(B)$.*

Proof. We present a proof for the more difficult case $c \in B_H$. For $c \in B_P$ the fact can be seen by analogous arguments. The geometric object corresponding to c is a hyperplane $h \in H$, parameterized by $\varepsilon^* := \varepsilon^*(B)$, for short: $h(\varepsilon^*) \in B_H(\varepsilon^*)$.

Suppose, the sphere $\mathcal{S}_0 := \mathcal{S}^*(B)$ has no point in common with $h(\varepsilon^*)$. This immediately contradicts optimality, if $|B_H| = 1$. Let $|B_H| \geq 2$.

Now, we consider $\mathcal{S}_1 := \mathcal{S}^*(B \setminus \{c\})$ and distinguish two cases. The first case applies, if one of both spheres is contained inside the other. Since $w(B\setminus\{c\}) < w(B)$ holds, the both spheres have to be distinct. The sphere $\mathcal{S}_1$ proves to be a better sphere for the constraints of B, if we have $\mathcal{S}_1 \subset \mathcal{S}_0$. If $\mathcal{S}_0 \subset \mathcal{S}_1$ holds, the sphere $\mathcal{S}_0$ must be better than $\mathcal{S}_1$ for the constraints of $(B \setminus \{c\})$. Both results contradict optimality.

For the second case, consider $\mathcal{S}_\lambda := \mathrm{comb}_\lambda(\mathcal{S}_0, \mathcal{S}_1)$ which describes a continuous movement from $\mathcal{S}_0$ to $\mathcal{S}_1$, determined by λ. Hence, there must exist a $\lambda' \in (0;1)$, such that $h^-(\varepsilon^*)$ contains $\mathcal{S}_{\lambda'}$, but $h(\varepsilon^*)$ does not touch $\partial\mathcal{S}_{\lambda'}$. Observe, that the situation here is the same as in the proof of theorem 2. None of the spheres $\mathcal{S}_0, \mathcal{S}_1$ is equal to $\mathrm{conv}(\mathcal{S}_0, \mathcal{S}_1)$. We can apply lemma 1 in the same way to ensure the feasibility of $\mathcal{S}_{\lambda'}$ for B. We see, that no hyperplane of $B_H(\varepsilon^*)$ supports $\mathcal{S}_{\lambda'}$, which contradicts optimality. □

Theorem 4. *Let the pair (C, w) be defined as above. Then $\dim(C, w) \leq 2(d+1)$ holds.*

The following proof consists of a new logic/geometric argument. A similar, but more simple statement is proven in WELZL[8] by a perturbation argument whose application is not obviously correct for our case.

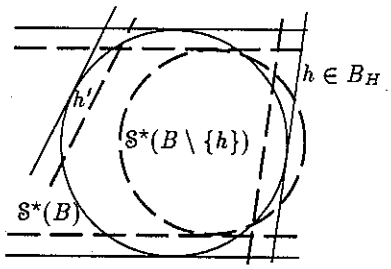
Proof. Suppose, there exists a basis $B \subseteq C$ with $|B| > 2(d+1)$. Then $|B_H| \geq d+2$ or $|B_P| \geq d+2$ holds. We take a look at the first case; the second is simplier and can be proven by similar arguments. Due to the cardinality of B_H, the values of all sets considered in this proof are defined.

First we show, that there exists a set $A \subseteq B$ with the property

$$\begin{aligned} &\forall h \in A\ \exists h' \in A \setminus \{h\} : \\ &w(B \setminus \{h\}) = w(B \setminus \{h, h'\}) > -\infty \end{aligned} \tag{3}$$

We use this property in the following way to achieve a contradiction: Argumenting with pigeon-hole-principle, we can claim, that there exists a constraint $h \in A$ whose corresponding $h' \in A \setminus \{h\}$ has the property $w(B \setminus \{h\}) \geq w(B \setminus \{h'\})$. (For an indirect proof of this, one may imagine the set A as the vertices of a finite graph and the correlations '$h \rightarrow h'$' as directed edges. Since every vertex has a leaving edge, there must be a cycle.) Using statement (3), we have $w(B \setminus \{h\}) = w(B \setminus \{h, h'\}) \geq w(B \setminus \{h'\})$. By *monotonicity*, this implies $w(B \setminus \{h, h'\}) = w(B \setminus \{h'\})$. Hence, we can examine both sets and the constraint h' for the condition of *locality*. We see, that it is not fulfilled, because h' is extreme in B, but not in $(B \setminus \{h\})$. We have a contradicition.

This corresponds to the logical part of the proof; statement (3) shall be proven geometrically. We set $\varepsilon^* := \varepsilon^*(B)$ and consider an arbitrary $h \in B_H$. Since B is a basis, we know by lemma 2, that $h(\varepsilon^*)$ supports $\mathcal{S}^*(B)$. The sphere $\mathcal{S}^*(B \setminus \{h\})$ must intersect $h(\varepsilon^*)$, as can be seen by a distinction of cases similar to the proof of lemma 2.



We assume, that all normals of the hyperplanes in B_H are in general position, i.e. interpreted as points, no $(d+1)$ of them are allowed to lie in a single hyperplane. Then we can show, that there exists a hyperplane $h' \in (B \setminus \{h\})$, parameterized by $\varepsilon^*(B \setminus \{h\})$, that does not support $\mathcal{S}^*(B \setminus \{h\})$. (To see this, we can suppose the converse. Consider the unique sphere that is supported by $(d+1)$ hyperplanes. If all hyperplanes are translated by the same distance (that can also be 0) in the reverse direction of their normals, they define a new, but concentric sphere that cannot intersect $h(\varepsilon^*)$. Explicitly, this can be shown by arguments of linear algebra.) Lemma 2 implies, that h' is not extreme for $(B \setminus \{h\})$. This proves (3) with $A := B_H$.

If the hyperplanes of B_H do not have the property of general position, as defined above, we are able to prove the existence of a set $A \subseteq B_H$, with $|A| \geq 4$, as needed for (3), by induction over d. We omit further details here. □

Remark 3. We cannot prove basis-regularity. Similar problems as the 1-center problem are not basis-regular, too.

In the planar case we can even improve our result and show $\dim(C, w) \leq 4$ by using fundamental properties of circles. We conjecture, that $\dim(C, w) \leq d+2$

holds for all d. This is justified by the following observation: A sphere is uniquely defined by k points and l *parameterized* hyperplanes, with $l + k = d + 2$, iff all objects are in general position. We can procede as before to establish an upper bound of $(d+2)$ for the combinatorial dimension under those special assumptions.

4.4 Algorithmic issues

An algorithm for solving a LP-type problem is given by SHARIR and WELZL[5]. The input for this randomized and recursive algorithm is the set of constraints C and an arbitrary initial basis A. The output is a basis for C.

It is not difficult to find an initial basis A, even with $\varepsilon^*(A) > 0$. We only need to find a pair of adjacent and non-coplanar facets of $\mathcal{P}$, defined by hyperplanes $h_1, h_2 \in H$. Let $\mathbf{p}_1 \in P$ be a vertex that lies in both facets, and $\mathbf{p}_2 \in P$ be an arbitrary further vertex. We set $B := \{h_1, h_2, \mathbf{p}_1, \mathbf{p}_2\}$ and use $A := \texttt{basis}(B)$ as the required basis.

Obviously, both primitive operations can be implemented in constant time, for our problem. A result of SHARIR and WELZL[5] helps us to establish the following, summerizing result:

Corollary 1. Let $\mathcal{P} \subset \mathbb{R}^d$ be a polytope and N be the size of its complexity of description. Problems 1 and 2 can be solved in expected time $O(2^{2d}N)$.

5 Extensions to ellipsoids

In general, ellipsoids are very economical approximations to polytopes, with regard to the complexity of description. For applications to collision-detection we are interested in finding an enclosing ellipsoid $\mathcal{E}^*$ that minimizes the Hausdorff-distance. Until now, notions like *smallest enclosing* or *minimum spanning* ellipsoids denoted ellipsoids with minimal volume. (See [8], [2].) We only want to mention our results shortly; for details the reader is again refered to the full paper.

Summerizing, the following holds: Restricting us to the planar case, we can solve the problem by the methods of section 4. We observe, that ellipses can touch the spherical parts of the parallel-body, if the Euclidean norm underlies the Hausdorff-distance, for example. For reasons of complexity, we neglect this detail.

We define a LP-type problem (C, w) in the same way as before. Again we are able to prove uniqueness.

Theorem 5. *Let $\mathcal{C} \subset \mathbb{R}^2$ be a planar convex body. The optimal ellipse $\mathcal{S}^*$ that minimizes $\delta^H(\mathcal{S}^*, \mathcal{C})$, subject to $\mathcal{S}^* \supseteq \mathcal{C}$, is unique.*

To show this, we use techniques similar to the proof of theorem 3. We distinguish several cases according to the number of common points with common tangents on the boundaries of two optimal ellipses.

Finally, the combinatorial dimension must be shown to be restricted.

Theorem 6. *Let the pair (C, w) be defined as above. Then $\dim(C, w) \leq 8$ holds.*

This statement is proven in a completely different way. We study the planar subdivision generated by an ellipse and some supporting lines that correpond to a basis.

There are negative results concerning a straightforward generalization to higher dimensions:

Remark 4. There exist convex bodies that admit unfinitely many Hausdorff-minimal ellipsoids.

Acknowledgments

The author would like to thank Günter Hotz for his encouragement and support, and Frank Follert and Elmar Schömer for several helpful discussions and for reading preliminary versions of this paper.

References

[1] LE, V. B., AND LEE, D. T. Out-of-roundness problem revisited. *IEEE Trans. Pattern Anal. Mach. Intell. PAMI-13*, 3 (1991), 217–223.

[2] POST, M. J. Minimum spanning ellipsoids. In *Proc. 16th Annu. ACM Sympos. Theory Comput.* (1984), pp. 108–116.

[3] SAMET, H. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.

[4] SEIDEL, R. Linear programming and convex hulls made easy. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.* (1990), pp. 211–215.

[5] SHARIR, M., AND WELZL, E. A combinatorial bound for linear programming and related problems. In *Proc. 9th Sympos. Theoret. Aspects Comput. Sci.* (1992), vol. 577 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 569–579.

[6] SWANSON, K., LEE, D. T., AND WU, V. L. An optimal algorithm for roundness determination on convex polygons. *Comput. Geom. Theory Appl. 5* (1995), 225–235.

[7] T.BONNESEN, AND W.FENCHEL. *Theorie der konvexen Körper.* Springer-Verlag, 1934.

[8] WELZL, E. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*, H. Maurer, Ed., vol. 555 of *Lecture Notes in Computer Science.* Springer-Verlag, 1991, pp. 359–370.

On hardness of roundness calculation

Sergey P. Tarasov *
Computing Center of RAS, Vavilova 40, Moscow, Russia
e-mail: sergey@ccas.ru

Abstract

We prove that it is NP-hard to approximate within high enough (but polynomial in the input size) accuracy roundness (or minimal width annulus or equivalently minimal deviating sphere) of a rational pointset in $\mathbf{R}^n$. We establish some simple combinatorial properties of the optimal pointsets. As an application we discuss stability of the Voronoi diagram.

Key words: computational geometry, roundness, NP-hardness, the Voronoi diagram.

1 Introduction

The question of finding minimal width annulus of a finite pointset arise often enough in the context of computational geometry, see, e.g. [6, 7, 8]. Similar problem arise in the Voronoi diagram stability computations [9].

Let $U \subset \mathbf{R}^n$ be any finite set. Denote by $DEV_s(U)$, respectively by $DEV_h(U)$, the minimal distance in Hausdorff metric from U to the set $\hat{S}$ of all Euclidean spheres or respectively to the set $\hat{H}$ of all $(n-1)$- dimensional affine hyperplanes of $\mathbf{R}^n$. Formally $(s \in \hat{S}) \leftrightarrow (\exists c \in \mathbf{R}^n, r \in \mathbf{R} \mid s = (y \in \mathbf{R}^n : \|y - c\| = r))$, $DEV(U) = \inf_{s \in \hat{S}}\{\max_{u_i \in U} dist(u_i, s)\}$, where $\|\cdot\|$ is the Euclidean norm and $dist(u_i, s)$ denotes the minimal Euclidean distance from a point u_i to a sphere s. $DEV_h(U)$ is defined analogously. Note that by definition $DEV_h(U)$ equals half of the *width* $w(U)$ of U. Analogously $DEV_s(U)$ equals half of the *width annulus* or *roundness* $sw(U)$ of U, where roundness by definition equals minimal distance between a pair of concentric spheres one that contains U and the other does not.

While considering complexity issues of the problems like width or roundness computation it is standard to adopt *binary* or *Turing* model of computation in which the *input data* should be represented as a finite number of binary strings that are processed by some Turing machine. The *size of the input* is a total size of the strings involved. The *time* or *space* complexity of the algorithm are defined in terms of the time and space complexity of the corresponding Turing machine. In particular this enables to speak about *polynomial* (in the size of the input) algorithms.

It is proved in [10] that it is NP-hard to approximate with high enough polynomial in the input accuracy the width of a rational simplex, and hence, the problem of computing the width of a polytope even if it is a rational simplex is in general intractable.

In this note we reduce the problem of approximate computing within high enough polynomial in the input size accuracy of the roundness of a rational pointset to the problem of bounding the width of a rational simplex. Thus roundness computation is also in general intractable.

2 Main construction

Our geometric construction is very similar to that of [10, Theorem 5.5][1]. But instead of simplex used in [10] we consider bipyramid. Namely take an arbitrary nondegenerate $(n-1)$-dimensional rational simplex $T_{n-1} = \mathrm{conv}\{a_1, \ldots, a_n\}$ in

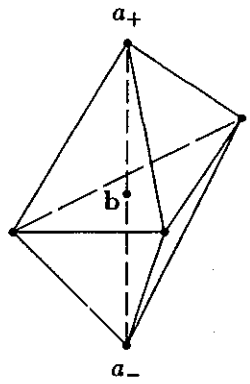


Fig. 1.

*Supported in part by grant RFFI 96-01-00662

[1] We are grateful to P.Gritzmann for the discussion of simplex width problem.

the subspace $L = \{x = (x_1, \ldots, x_n) \in \mathbf{R}^n | x_n = 0\}$. Let $b = 1/n(\sum_{i=1}^n a_i)$ be its barycenter and D its diameter in L. $L = \{x = (x_1 \ldots, x_n) \in \mathbf{R}^n | x_n = 0\}$. Let $b = 1/n(\sum_{i=1}^n a_i)$ be its barycenter and D its diameter in L. Note (see e.g. [10]) that the square of width of T_{n-1} in L and the square D^2 of the diameter of T_{n-1} are rational numbers not exceeding $32n^4l$, where l is the size of the input, i.e. total number of bits necessary to encode all rational vertices of the simplex. Set $a_\pm = b \pm he_n$ and $K = \{a_1, \ldots, a_n, a_+, a_-\}, P = \mathrm{conv}(K)$, where e_n is a standard basis vector and $h = 2^{161n^4l}$. Thus P is a bipyramid with apexes $a_\pm$ and the base T_{n-1}. We want to show that if it is possible to approximate roundness of K within accuracy greater than 2^{-64n^4l} then we can upper-bound $(n-1)$-dimensional width $w(T)$ of T_{n-1} and our result will follow from the following fact proved among all in [10].

Proposition. *The following problem is NP-complete:*

Instance: A positive integer n; $n+1$ rational points forming a simplex in $\mathbf{R}^n$; a positive rational λ.

Question: Is λ an upper bound for the square of the width of T?

Recalling that $(n-1)$-dimensional width of the simplex T_{n-1} in L is a rational number and its size is bounded by $32n^4l$ we conclude that in order to prove NP-hardness of roundness computation it is sufficient to prove the following inequality

$$sw^2(K) \le w^2(T_{n-1}) \le sw^2(K) + 2^{-64n^4l}. \quad (1)$$

The left inequality in (1) is evident as K (and P) is located between a pair of parallel hyperplanes that are orthogonal to L and supporting T_{n-1}. The distance between these planes is $w(T_{n-1})$.

Let prove the right inequality of (1). First assume that the center c of concentric spheres which specify width annulus is located at the infinity (i.e. corresponding concentric spheres are parallel hyperplanes). Let H_+ and H_- be parallel supporting hyperplanes of P whose distance equals the width of K. One of the vertices $a_\pm$ and one of the vertices of T_{n-1} should necessarily lie in one of the supporting hyperplanes. Thus if we denote by α the angle between the directional covector of the supporting hyperplane and the hyperplane $(x|x_n = 0)$ then $\tan\alpha \le \frac{D}{h}$. Therefore $sw(K) = w(K) \ge w(T_{n-1})\cos(\alpha)$ and $w^2(T_{n-1}) \le sw^2(K) + D^4h^{-2}$. Thus right inequality of (1) holds in this case. Let now the center c be located at a finite point. Place the origin at point b. We may assume that $\|ca_+\| \le \|ca_-\|$.

Let $_0$ be the projection of c on L. Let denote the lengths of the segments b_0 and c_0 by s and t, respectively.

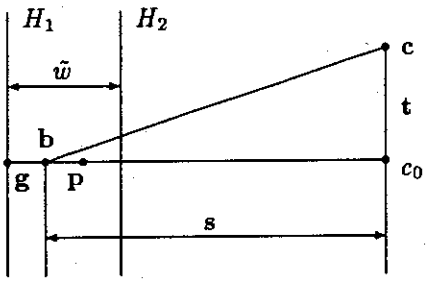


Fig. 2.

Let width annulus of K be realized by concentric spheres S_R and S_r, $sw(K) = R - r$, with the center at c and of the radii R and r, respectively. Let Q be two-dimensional plane passing through the points $\{c, a_+, a_-\}$. Let $g = S_R \cap Q \cap L$ and $p = S_r \cap Q \cap L$.

We claim that the angle $\beta = \angle cbc_0$ is small enough. Use the inequality $|ca_-| - |ca_+| \le w$. In our notation it can be rewritten as follows $\sqrt{(h+t)^2+s^2} - \sqrt{(h-t)^2+s^2} \le w$. This implies (as $|bc| \le R \le |bc| + D \quad r > \frac{h}{3}$) that $\frac{4th}{2(\sqrt{t^2+s^2}+D)} \le w$ and $\sin\beta = \frac{t}{|bc|} \le \frac{2t}{|bc|+D} \le \frac{w}{h}$ and $\tan\beta \le \frac{2w}{h}$. The same inequality holds for the angle $\angle cpc_0$.

It can be checked that $|gp|^2 - sw(K)^2 < 2sw(K)^2\tan\beta^2$ (as β and r^{-1} are small enough).

Now let H_1, H_2 be such a pair of orthogonal to the line gc_0 hyperplanes that $g \in H_1$ and H_2 is a supporting hyperplane to P and P lies between H_1 and H_2. By construction H_1 is also supporting hyperplane to P. Denote the distance between H_1 and H_2 by $\tilde{w}$. Let v be an arbitrary vertex of K in H_2. By the above v lies on the $(n-2)$-sphere S_ρ centered at c_0 and of the radius $\rho \ge r\cos(\angle cgc_0) \ge \frac{h}{4}$. Moreover the distance between v and the line gc_0 is $\le D$. Therefore $w(T_{n-1}) \le \tilde{w} \le |gp| + 2\frac{D^2}{\rho}$ $w^2(T_{n-1}) \le |gp|^2 + 32\frac{D^3}{h} \le sw^2(K) + 64\frac{D^3}{h}$ and the right inequality of (1) holds.

This proves the following theorem.

Theorem 1. *Approximate computation of the width annulus or roundness of a system of $n+2$ rational points in $\mathbf{R}^n$ within high enough (but polynomial in the size) accuracy is NP-hard*

3 Combinatorial properties of roundness[1]

It is clear that width and roundness computation of the finite system of points $U = \{u_1, \ldots, u_l\} \subset \mathbf{R}^n$ can be reduced to solving some finite-dimensional

[1] This section is written jointly with M.N.Vyalyi

optimization problems. Some simple combinatorial properties of the width of the polytope $P = convex(U)$ in an arbitrary finite-dimensional normed space (*Minkowski space*) are established in [11]

Proposition 2 [11]. *Let H_+ and H_- be parallel supporting hyperplanes of $P = convex(U)$ whose distance is equal to the width of P (we assume that $w(P) > 0$). Let $P_+ = P \cap H_+$ and $P_- = P \cap H_-$. Then $dim P_+ + dim P_- \geq n-1$, with $dim P_+ = dim P_- = n-1$ when P is centrally symmetric. Let H denote the hyperplane that is parallel to and equidistant from H_+ and H_-. Denote by $P^+, P^- \subset H$ the sets lying at the distance $w(P)/2$ from P^+ and P^- respectively. Then the sets P^+ and P^- are not strictly separated in H. Moreover, if a unique supporting hyperplane passes through any boundary point of the unit ball of the space involved (i.e. the ball is* smooth*) then P^+ and P^- are not even weakly separated in H.*

We shall show that similar characterization of roundness can be obtained in the Euclidean case.

Clearly determining $DEV_s(U)$ is equivalent to finding global infimum of the function $\delta(y) = sup_{u\in U}\|y-u\| - inf_{u\in U}\|y-u\|$. Denote this infimum by Δ. By definition $DEV_s(U) = \Delta/2$.

Let the infimum of $DEV_s(U)$ be attained at a finite point c. Then c is the center of the optimal sphere $S^*(C^*, r^*)$. The radius may be recovered by the formula $r^* = (max_{u\in U}\|C^*-u\| + min_{u\in U}\|C^*-u\|)/2$.

If the infimum of $DEV_s(U)$ is attained at the infinity then the optimal sphere is a hyperplane and $sw(U) = w(U)$.

If x is a point and S is any sphere then the closest to x point on S is called a *projection* and is denoted by x^S. Thus $x^S = Argmin_{v\in S}(\|x-v\|)$. The projection map is one-to-one except for the center of the sphere involved.

If the infimum of $DEV_s(U)$ is attained at a finite point then let project the extremum points on the optimal sphere. We distinguish two subsets:

$$A_{in} = \{v | v = u^{S^*}, u \in Argmin(\|C^* - u\|)\}$$

$$A^{out} = \{v | v = u^{S^*}, u \in Argmax(\|C^* - u\|)\}$$

If the infimum of $DEV_s(U)$ is attained at the infinity then let define the subsets A_{in} and A^{out} using notation from the Proposition 2 as follows: $A_{in} = U \cap H_- + \mathbf{n}w(U)$ and $A^{out} = U \cap H_+ - \mathbf{n}w(U)$, here $\mathbf{n}$ – is a unit normal to H^+ chosen in such a way that $A_{in}, A^{out} \subset H = \frac{H_+ + H_-}{2}$.

Theorem 2. *Let $sw(U) > 0$. Then*

(i) A_{in} and A^{out} are not linear separated.

(ii) $|A_{in}| + |A^{out}| \geq n+2$.

Proof. Let prove *(i)*. In view of the Proposition 2 it is enough to consider the case where infimum of $DEV_s(U)$ is attained at finite point. Assume on the contrary that there exists a separating hyperplane $Q = \{x | qx = q_0\}$ such that $qx|_{A_{in}} \leq q_0$ and $qx|_{A^{out}} > q_0$. Denote by s $(n-2)$–sphere formed by intersection of Q and S, so that $s = S \cap Q$. Now for a positive ε small enough the sphere centered at the point $C^* + \varepsilon q$ and passing through the sphere s has smaller deviation from the set U. A contradiction.

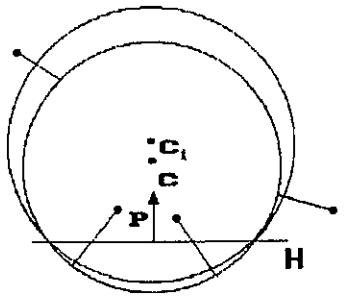


Fig. 3.

Now let prove *(ii)*. First let consider finite case. Let $L = affine\ hull(A_{in}, A^{out})$. Now if $|A_{in}| + |A^{out}| < n+2$ then it immediately follows from *(i)* that $\dim L < n$ (otherwise, sets A_{in} and A^{out} can be linear separated as they form simplex). If necessary arbitrarily extend L to hyperplane (i.e. $\dim L = n-1$). If $C - C_1 \neq 0$ then let $p = C - C_1$, where C_1 is a center of the $(n-2)$-sphere $L \cap S$. Otherwise, set $p = \tilde{p}$, where $\tilde{p}$ is orthogonal to L,

Proceeding as in *(i)* it is easy to check that for a positive ε small enough the sphere centered at the point $C^* + \varepsilon p$ and passing through the sphere $L \cap S$ has smaller deviation from the set U. A contradiction.

Let now the optimal sphere be hyperplane.

The preimages of points in A_{in} and A^{out} under projection map are not contained in any hyperplane and form a n-dimensional simplex A (otherwise A_{in} and A^{out} would be linear separated in H that contradicts Proposition 2). Let S_A be the sphere circumscribed around A and centered at O.

Using notation of the Proposition 2 let $S_+ = S_A \cap H_+$ and $S_- = S_A \cap H_-$ be $(n-2)$-spheres – ressections of S_A by the hyperplanes H_+ H_- of the radii r_+ and r_-, respectively. Let consider a pair of concentric spheres passing respectively through $(n-2)$-spheres S_+ and S_- centered at the point $\tilde{O} = O + \lambda\mathbf{n}$. Here $\mathbf{n}$

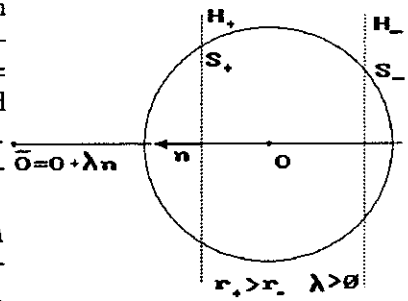


Fig. 4.

is a unit normal to the hyperplane H_+; λ should have large enough absolute value and should have such a sign so that the distance from $\tilde{O}$ to the $(n-2)$-sphere-section S_+ or S_- with *smaller* radius were *greater* (see Fig. 4). Now direct calculation shows that if the absolute value of λ is large enough then the set U is contained between the concentric spheres involved and the distance between these spheres is smaller than the distance between the hyperplanes H_+ and H_-. A contradiction. □

Corollary 1. *The inequalities* $|A_{in}| > 1$ *and* $|A^{out}| > 1$ *should hold.*

Proof. The finite case follows immediately from *(i)* of the previous theorem as any point on the sphere can be linear separated from any finite set. Infinite two-dimensional case was considered in [7] and similar to the analysis of the infinite case in *(ii)* above can be readily generalized to higher dimensions. □

Thus in the plane $\mathbf{R}^2$ an optimal circle (or infinite circle– line) minimizing deviation from any four points should either pass them in alternating fashion (i.e. the optimal circle pass through {in} and {out} points alternatingly) or pass through all of them thus demonstrating a pattern standard for Chebyshev-type approximation problems. This property leads to a straightforward algorithm for roundness computation of a quadruple of points in the plane. Partition the quadruple into pairs and draw mid-perpendiculars to pairs. If they cross at a point C take the circle centered at C of the radius equal to half the sum of distances from C to pairs. Otherwise (if mid-perpendiculars does not cross) take an infinite circle– line orthogonal to both bisectors and passing at equal distances from pairs.

Project points on the circle (or line) involved and check alternation condition. Repeat this procedure for all partitions of the quadruple into pairs and choose among all circles satisfying alternation condition the one with the minimal distance from the quadruple.

For example, for the quadruple of points that consists of the vertices a, b, c of a regular triangle and its barycenter there are three optimal circles of radius $\frac{\sqrt{7}+1}{\sqrt{48}}l$ centered at points $\frac{a+O}{2}, \frac{b+O}{2}, \frac{c+O}{2}$, respectively ($l$ is the edge length). The roundness of this configuration equals to $sw = \frac{\sqrt{7}-1}{\sqrt{12}}l$.

Note that Corollary 1 (and the analogous statement for the plane [7, Theorem 2]) is false for infinite U. For example, let $U = [(-1,0),(1,0)] \cup (0,\varepsilon) \subset \mathbf{R}^2$ consists of a segment and a point that is located near enough to the segment. Then the width and roundness of the configuration are equal to ε.

Concluding this section we note that Theorem 2 trivially results in a pseudopolynomial in the dimension (and polynomial when the dimension is fixed) algorithm for computation roundness of a configuration of (rational) points. Analogous procedure also exists by the Voronoi diagram arguments similar to stated in [6, 7, 8]. Polynomial solvability in fixed dimension also follows from effective variants of the well-known Tarskii algorithm (see, e.g. [12]).

As it has been already mentioned width and roundness computations arise among all in the framework of stability of the Voronoi diagram which we discuss in the next section.

4 Stability radius of the Voronoi diagram in the Euclidean case

Let $P = \{p_1, \ldots, p_k\}$ be a set of sites in n-dimensional Euclidean space $\mathbf{R}^n$. Recall that the *Voronoi diagram* V of P (see, e.g. [1]) is the partition of $\mathbf{R}^n$ into k so called Dirichlet regions $P_1, \ldots, P_k$, where $P_i = \{x \in \mathbf{R}^n : \|x - p_i\| \leq \|x - p_j\|, j = 1, \ldots, k\}$, and $\|\cdot\|$ denotes the usual Euclidean norm in $\mathbf{R}^n$.

All definitions below are borrowed from [9], where some general results are obtained for the stability of Voronoi diagrams not only for the Euclidean norm, but also for arbitrary star-shaped and regular in the sense of [2] distance functions. Some stability questions for plane geometric objects are discussed among all in [5]. In particular, [5] suggests a method for computing the *tolerance* of plane Delaunay triangulations, a quantity similar to our stability radius.

We first define the *intersection hypergraph* $H(P)$ of the Voronoi diagram V. The vertices of $H(p)$ are the Dirichlet regions $P_1, \ldots, P_k$, and the hyperedges of $H(p)$ are the emptyset and all those collections of the regions P_i that have nonempty intersection in $\mathbf{R}^n$. The hypergraph $H(P)$ is thus by definition a simplicial complex.

Two Voronoi diagrams are called *equivalent* if they have identical intersection hypergraphs.

Let $\mathcal{M} = \underbrace{\mathbf{R}^n \times \mathbf{R}^n \times \ldots \times \mathbf{R}^n}_{k \text{ times}}$ be the *configuration space* of the diagram.

We shall assume throughout this section that the distance between two points $\mathcal{P} = (p_1, \ldots, p_k)$ and $\mathcal{Q} = (q_1, \ldots, q_k)$ of $\mathcal{M}$ equals to $dist(\mathcal{P}, \mathcal{Q}) = \max_{i=1,\ldots,k}\{\|p_i - q_i\|\}$. In other words we assume that if the sites are perturbed independently then

the extent of a particular perturbation is measured by the maximum Euclidean distance between the original and the deviated positions of sites in the base space.

A diagram V (treated as a point $\mathcal{P} = (p_1, \ldots, p_k)$ in the configuration space) is called *stable* [9] if all diagrams in some neighborhood of $\mathcal{P}$ in $\mathcal{M}$ are equivalent.

The infimum of the distance in the configuration space from a point $\mathcal{P}$ to the set of the nonstable configurations is called *stability radius* of the Voronoi diagram V and is denoted by $Stab(V)$.

As usual let denote by $B(C, r)$ and $S(C, r)$, respectively a ball and a sphere of radius r and having center C. Now we'll use the notion of the *empty ball* introduced by Delaunay (a ball $B(C, r) \subset \mathbf{R}^n$ is called *empty* if it has no sites inside).

A set of sites lying on the boundary of some empty ball is called a *combinatorial empty sphere.*

Denote by $\Gamma(P)$ the hypergraph of combinatorial empty spheres. It has sites as vertices and combinatorial empty spheres as edges.

By *Delaunay duality* hypergraphs $H(P)$ and $\Gamma(P)$ are isomorphic. Thus we immediately conclude that diagram is stable iff all its subconfigurations– combinatorial empty spheres– are stable and no new combinatorial empty spheres arise in some neighborhood of $\mathcal{P}$ in the configuration space.

So to test the stability of the diagram we should single out subconfigurations of sites that are not combinatorial empty spheres but fall on the boundary of some empty ball after some arbitrary small perturbations. In a recent paper [9] the configurations not lying on the boundary of any ball are labelled as *antispheres.* Moreover we call an antisphere *unstable* if in a no matter how small neighborhood of it in its configuration space there is a combinatorial empty sphere. It was proved in [9] that for a general class of distance functions unstable antispheres are strongly related to the boundary of some *supporting cone* of the unit ball of the space involved. Thus in our Euclidean case all antispheres are simply subsets of sites lying on particular facets of the convex hull of the sites of the diagram. This observation immediately follows from simple lemma.

Lemma 1. *Let U be a configuration of sites not belonging to any sphere and let $U_i \subset S(C_i, r_i)$ be a sequence of configurations that are arbitrarily close to U in its configuration space as i tends to infinity.*

Then $r_i \to \infty$ when $i \to \infty$.

So a criterion for the stability of the Voronoi diagram in the Euclidean space $\mathbf{R}^n$ may be formulated as follows.

Let $l =$ dim(affine hull (P)). If the set of sites is not full dimensional (i.e. if $l < n$) then clearly the Voronoi diagram is stable iff the sites form a nondegenerate simplex or $l = |P| - 1$.

Otherwise two conditions should hold.

1. Geometrically dual to the Voronoi diagram Delaunay complex should be a triangulation of the convex hull of the sites of the diagram. (Of course this rather obvious condition has been widely used already by Voronoi himself in the theory of types of quadratic forms [3]. Delaunay made it geometrically transparent via the theory of "la sphere vide", see, e.g. [4].)

2. Each subset of sites belonging to the same facet of the convex hull of the diagram should be affinely independent.

Therefore to test stability of the diagram or more generally to calculate its stability radius we should take minimum of the stability radii of all combinatorial spheres and antispheres (that are immediately at hand as by product of the diagram construction). Now we specify the sets of combinatorial spheres and antispheres involved a bit more precisely.

Let K_n be the complex of all n–dimensional simplices of Delaunay triangulation. Let K_{n-1} be the complex of all $(n-1)$–dimensional *boundary* simplices of the convex hull of the sites of the diagram. For $k \in K_n$ let $U_k = \{u_i\}$ be the set of neighboring sites from the star of k in the triangulation. For $k \in K_{n-1}$ let $U_k = \{u_i\}$ be the union of neighboring sites from the star of k in the boundary triangulation induced by Delaunay triangulation and the site that complements k to the unique n–dimensional simplex of Delaunay triangulation.

Theorem 3. *The following formula holds for the stability radius of the diagram:* $Stab(V) = min\{min_{k \in K_n} min_{u_i \in U_k} DEV_s(u_i \cup k), min_{k \in K_{n-1}} min_{u_i \in U_k} DEV_h(u_i \cup k)\}$.

Proof. Note that in general we should test stability of maximal by inclusion spheres and minimal by inclusion antispheres only.

Let $ax \leq b$ be any linear inequality defining facet of the convex hull of the sites of the diagram. The corresponding supporting hyperplane $(x \in \mathbf{R^n} | ax = b)$ and the complementary closed halfspace $(x \in \mathbf{R^n} | ax \geq b)$ are called an *infinite empty sphere* and an *infinite empty ball*, respectively.

Now let consider the set of all *geometric* empty spheres circumscribed around n-dimensional simplices of the Delaunay triangulation and let enlarge it by the set of all infinite empty spheres. By stan-

dard argument for the Delaunay triangulation the surface of any geometric empty sphere is completely *covered* by neighboring empty balls (including infinite ones).

This enlarged set of all geometric empty spheres varies under perturbation of sites but as long as diagram remains stable no new geometric empty spheres can arise or disappear. Thus geometrically diagram looses stability iff some pair of geometric empty spheres coincide. This means that one extra site should fall on the boundary of the first geometric empty sphere involved or vice versa. Due to covering property of empty balls this extra site should belong to one of the neighboring geometric empty spheres. If two finite geometric spheres coincide then with necessity this extra site should be one of the neighboring sites from the star of the simplex involved in the Delaunay triangulation. Let two infinite spheres coincide. Denote by F the simplicial facet that generates one of the infinite spheres involved. Then extra site should belong either to the set of neighboring sites from the star of F in simplicial boundary complex of the convex hull of the sites of the diagram or be the site that complements F to the unique n–dimensional simplex in Delaunay triangulation.

Combining these observation together we get the resulting formula for the stability radius. □

So to compute the stability radius we come up against two optimization problems that we state slightly more generally. We put specifications of our problems in square brackets.

1. To find a [finite] sphere s having minimal deviation $DEV_s(U)$ from a given set U, i.e. to find width annulus [of $n+2$ nonconspheric points not belonging to any affine hyperplane].

2. To find a $(n-1)$–dimensional hyperplane h having minimal deviation $DEV_h(U)$ from a given set U, i.e. to find width [of $n+1$ points in general position].

It is well known that number of simplices in Delaunay triangulation is $O(k^{\lceil n/2 \rceil})$ and thus is polynomial if the dimension of the space involved is fixed. Hence, it follows from the discussion above that if the dimension is fixed then the stability radius of the Voronoi diagram can be approximated with arbitrary accuracy in polynomial time and, in particular, in linear time in the plane.

But it also follows that in general approximation of the stability radius with arbitrary polynomial in the input size accuracy is NP–hard a problem even for a rational simplex.

References

[1] F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure, *ACM Computing Surveys*, **23** , No. 3, (1991), 345–405.

[2] F. Clarke. *Optimization and nonsmooth analysis*, John Wiley&Sons, N.-Y., 1983.

[3] G. F. Voronoi. Recherches sur les paralleloédres primitifs *Crelle J.*, **134**, 198–287, (1908); **136**, 67–179 (1909).

[4] B. N. Delaunay. *Petersburg number theory school*, Academy of Sciences, M.-L., USSR, 1947.

[5] M.Abellanas, F. Hurtado and P. Ramos. Redrawing a graph within a geometric tolerance, Graph Drawing,eds. R.Tamassia and I.Tollis, in *Graph Drawing. Lecture Notes in Computer science.* Springer, Berlin-Heidelberg 1995.

[6] P. Agarwal, M. Sharir. Efficient randomized algorithms for some geometric optimization problems, Proc. 11th Annu. Symp. Comput. Geom., 1995, 326–335.

[7] M. Smid, R. Janardan. On the width and roundness of a set of points in the plane, Proc. 7th Annu. Canad. Conf. Comput. Geom., 1995.

[8] K. Swanson. An optimal algorithm for roundness determination of convex polygons. Proc. 3rd. WADS, Lecture Notes in Computer Science, vol. 709, Springer-Verlag, Berlin, 1993, 601–609.

[9] M. Vyalyi, E. Gordeev and S. Tarasov. The stability of the Voronoi diagram, *Comp. Maths Math. Phys.*, **36**, 1996, 405–414.

[10] P. Gritzmann, V. Klee. Computational complexity of inner and outer j-radii of polytopes in finite-dimensional normed spaces, *Math.Prog.*, **59**, 1993, 163–213.

[11] P. Gritzmann, V. Klee. Inner and outer j-radii of convex bodies in finite-dimensional normed spaces. Discrete Computational Geometry, **7**, 1992, 255–280

[12] U. Faigle, W. Kern and M. Streng. Note on the computational complexity of j-radii of polytopes in $\mathbf{R}^n$. *Math.Prog.*, **73**, 1996, N1.

Applied Geometry and Computer Graphics

Eugene Fiume
Department of Computer Science
University of Toronto
10 King's College Circle
Toronto, Canada
M5S 3G4
elf@dgp.toronto.edu

and
Alias|Wavefront Inc.
110 Richmond St. East
Toronto, Canada
M5C 3P1
elf@aw.sgi.com

Abstract

It has become a priority in computational geometry to strive for greater impact by searching for ways of applying techniques from computational geometry to "real" computation. Likewise, some problems in computer graphics can greatly benefit from applied computational geometry. However, I argue that this important effort requires computational geometers to first acquire a clear understanding of the pragmatics and culture of the application. I will illustrate this point through the use of story about a classic geometric visibility algorithm. I will then enumerate several problem areas within computer graphics that could significantly benefit from collaboration with computational geometers.

1 Some History

It is difficult to speak of a "classical" algorithm in computer graphics, for the earliest seminal work in our field is that of Ivan Sutherland on *Sketchpad* in 1963, less than 35 years ago. But few would deny that Gary Watkins' dissertation, the shortest Ph.D. thesis I have seen in computer graphics, is a classical algorithm. Often cited as a 1969 technical report from the University of Utah, the birthplace of computer graphics, only a privileged few have seen the original work. Most of us will have to be content with the standard presentation found in good introductory graphics texts. To their great discredit, some texts entirely omit a discussion of this algorithm, since it has sadly fallen into disuse. Nowadays, this kind of material seems to be better covered in computational geometry texts.

The thesis described a beautiful visibility algorithm that would now be classified as being *line sweep*. In computer graphics, we call it a *scanline algorithm*, because the sweeping line corresponds to a *scanline*, or row, of image pixels. The fundamental data structures are: an *active edge table* (AET), which maintains the edges of polygons, sorted from left to right, that overlap with a given scanline of an image to be rendered; and a *scanline bucket*, which gives the list of polygons that enter at a given scanline. Upon imposing a scanning direction of, say, top to bottom, a three-dimensional polygonal input scene could be processed in one pass, and the AET could be incrementally adjusted in going from one scanline to its successor. In particular, if y denotes a given (integral) scanline, and x denotes the (floating point) position of an edge along that scanline, then for the next scanline given by $y - 1$, x could be incrementally adjusted, or the edge could be removed from the AET if that scanline has passed both of its endpoints.

This algorithm had several shortcomings, many of which were corrected in later work: it could not handle polygons that interpenetrated, it could choke on nonplanar polygons,[1] it didn't work for concave polygons, and it could have some severe robustness problems with nearly horizontal edges, sliver polygons, and other corner effects. Nevertheless, this algorithm changed the way computer graphics was done: it made clear the importance of efficient, applied geometric algorithms to computer graphics.

Naturally, many people wanted to understand the computational complexity of the algorithm. A sim-

[1] Because polygonal representations are often derived from polynomial surfaces, we cannot guarantee the planarity of quadrilaterals in computer graphics. Even on our worst days, however, we can manage to give users planar triangles!

ple analysis suggests that for n input k-gons with k constant, there could be $\Theta(n^2)$ fragmented polygonal spans overlapping with an edge; the requirement to sort these edges would result in an $\Omega(n^2 \log n)$ lower bound. Certainly this would be the common wisdom from computational geometry. Watkins' algorithm guarantees that each pixel in the image will be visited exactly once. Depending on whether the output resolution is seen as constant or variable, one may wish to factor the pixel fill cost into the overall complexity of the algorithm.

Watkins was pragmatic. He saw that polygons making up the input scene generally had significant spatial *coherence*, and he capitalised on that.[2] He observed that in updating the AET, generally a small number of new edges were being added to an already sorted list, or that there would be a small number of interchanges. He thus heuristically employed exchange-sort and *incremental updates* to the AET, usually giving a practical sublinear scanline update complexity. Subsequent empirical analysis showed that configurations achieving the worst case of $O(n^2)$ polygons are rare, and that having much fewer than $O(n)$ polygons in the AET was common. Thus we expect that Watkins' algorithm usually runs in close to linear time.

Elegance, speed, quality. Together with another lovely algorithm by his contemporary, Warnock, Watkins' algorithm had all the makings of forever being a dominant algorithm in computer graphics.

But life is not fair. In short order, Watkins' and Warnock's algorithms were summarily dispatched by a brute force, inelegant, apparently slow, easy-to-code algorithm called the *depth*-buffer algorithm. More commonly, it is called the z-buffer algorithm, because z corresponds to a depth value after a change of basis to a left-handed co-ordinate system and after perspective transformation. It is depicted in Algorithm 1.

[2]The term *coherence* in computer graphics is a heuristic catch-all, referring to the idea that even in complex environments, there tends to be significant correlation within many of the quantities we compute or measure. For example, we would expect illumination, shadows, depths, and visibility to vary only by small amounts as we compute nearby pixel values. Exceptions are relatively infrequent. This is reflected by the view that an array of *pixels*, which are piecewise constant colour values, represents a sufficiently dense sampling of our virtual environment.

```
First, we initialise.
∀ pixels (i,j) ∈ image
    depth[i,j] ← ∞
    colour[i,j] ← background
Now resolve image space visibility.
∀ polygons P
    ∀ pixels (i,j) ∈ P
        if Z_P(i,j) < depth[i,j]
            depth[i,j] ← Z_P(i,j)
            colour[i,j] ← Shade_P(i,j)
```

Algorithm 1: The depth buffer algorithm.

In practice, the point-in-polygon problem is reduced to a constant-time linear interpolation. Similarly, the computation of depths and shading are based on linear interpolation.

Why did the z-buffer algorithm supplant Watkins? There are several reasons. First, the z-buffer algorithm is exceptionally simple, with fixed bounds on output complexity based mostly on image resolution (see below). This makes it ideal for hardware. Second, as scene complexity (as measured by the number of polygons) increases, their projected size on the image plane tends to decrease.[3] Thus the actual number of *pixels* filled tends to be independent of the number of input polygons. Third, much of the computation can be done in low precision. Fourth, it is possible to extend the z-buffer algorithm to perform anti-aliasing (which was always possible with Watkins' algorithm), a breakthrough that permitted high quality hardware renderings to be produced. Fifth, interpenetrating polygons are not problematic: the algorithm only depends on resolving the depth function of a polygon at each pixel. Often overlooked is that with high *depth complexity*, i.e., when many polygons having different depth values overlap with a pixel, numerous expensive shading computations are wasted for invisible polygons (cf. the last lines of the above algorithm). That is, unlike Watkin's algorithm, the number of pixels visited is not constant (although the number of distinct pixels is constant).

By the time 3D graphics standards such as OpenGL started appearing in the 80's, elegant rendering algorithms based on Watkins and Warnock es-

[3]In fact, polygons are often much smaller than a pixel in size.

sentially disappeared. By a strange twist of irony, Watkins' algorithm was in fact implemented in hardware in the 1970's. James Clarke, who went on to design the core graphics architecture of the SGI graphics workstations, learned about graphics hardware by studying the hardware implementation of Watkins' algorithm at Utah. Clarke was the inventor of the Geometry Engine and champion of the z-buffer approach. But there is a happy ending, as we shall see.

There are a couple of points I would like to make about this folk story. The first one is that in computer graphics, we cannot assume that using algorithms of lower asymptotic complexity will give us better real performance. The use of the asymptotically-slower exchange sort in Watkins' algorithm is important when we recall the performance of quick-sort, for example, on nearly sorted input. The second is that the best or most elegant algorithm doesn't always finish first. In computer graphics, models of computation are certain to change on us, which may well change the objective definition of *best*. Indeed, the constant need for greater throughput drives us to explore different models of computation and to employ algorithms that better match these models.

I would like to drive home the point of asymptotic complexity. One of the most interesting geometric problems in computer graphics is the computation of shadows in polyhedral environments arising from area light sources. Several years ago, George Drettakis and I were involved in the design of an intricate shadow algorithm. It consisted of a sequence of geometric operations. The composition of the worst cases of the constituent algorithms we employed resulted in an overall multiplicative worst case of the components, giving an absurd $O(n^{18})$ upper bound. In an area where $O(n^2)$ algorithms are considered functionally intractable, this would be a disaster. However, we were never able to generate a worst case that came anywhere close to worst case, and in fact our algorithm stubbornly exhibited strongly subquadratic performance. James Stewart and Sherif Ghali, also from the graphics lab at the University of Toronto, concurrently developed another shadow algorithm with better worst case performance than ours; again their implementation so clearly outperforms the worst case analysis that it gives one pause as to the real value of such analysis for practical algorithms. I am certainly not the first to have spoken of this credibility gap.

An obvious analogy here can be drawn to scientific computation. The error bounds for numerical algorithms are derived so as to apply to a broad function space, including pathological functions that do not readily occur in practice. The extent to which an error bound is dominated by the behaviour of a numerical algorithm on rare pathological functions determines the relevance of that bound. An identical argument can be made regarding the relevance both of specific geometric algorithms and of geometric complexity analysis to computer graphics.

2 Finding a Middle Ground

In numerical analysis, it is no easier to define objectively the "relevant functions" of an application as a restriction on, say, the class of L^2-integrable functions, than it is for us in computer graphics to define a "typical scene". This complicates the transfer of innovative algorithms in computational geometry to computer graphics. One might get lucky and find that the 2-D visibility problem for, say, star polygons is just the right class of input for some graphics application; more likely, the domain of typical inputs we will want in graphics will cut raggedly and non-inclusively through cleanly specified classes of input. While computational geometers were right to follow research trajectories that focused on specific classes of polygons, there is little to suggest that the resulting algorithms on these classes are necessarily relevant to computer graphics.

The match between computer graphics and computational geometry is not perfect. If we accept the argument that geometric algorithms must be better adapted to problems to computer graphics, and if we agree that one cannot in general define the class of typical graphical inputs, then it follows that applying geometric algorithms to computer graphics requires becoming aware of the "culture" of computer graphics. This is both a social and technical process. Without such a cultural awareness, the synergies between our communities will be suboptimal. Naturally, the better situation is profound mutual awareness, but I will focus on one-way awareness for now.

3 The Computer Graphics Culture

It's never easy to define a "culture", and I will leave it to some strand of anthropology to make it precise, but there are some properties of a computer graphics culture that are fairly easy to isolate.

Having an eye for images. The ability to look critically and analytically at images seems like a self-evident property of the graphics culture. It is rather more of a skill than first appears. We all have, in fact, a tendency to accept images too readily. The analysis of computer-generated images and the ability to account for artifacts in them is the most important aspect of our craft. It won't do simply to say "It doesn't look right" without a reason. Computer graphics people love to practise a form of visual forensics by tracing image artifacts back to the algorithms that were used to compute the image. Recall, for example, that Watkins' algorithm did not handle interpenetrating polygons. The z-buffer algorithm blindly handles them by accident; because the regular z-buffer algorithm is only aware of the original edges, not induced edges given by interpenetration, the algorithm will happily anti-alias the original edges, but not the interpenetrations. These signatures are very evident. Furthermore, while it is possible to treat computer graphics more abstractly as a mathematical science, without a passion for making and assessing images, it is difficult to believe that the satisfaction of getting a particular visual effect right will offset the intimidating effort required to achieve it. Computer graphics people know and appreciate the difficulty of creating images.

Knowing the "pragmatics" of error. A computer graphics person knows that, whether through incomplete models, imprecise algorithms, or plain and simple hacks, significant numerical errors will always exist in our results. We are attuned to understanding the nature of errors and we try to divert error into the least evident visual manifestation possible. Some of this knowledge is folklore, while some is based on reasonably good science. We know that broadband noise is generally less perceptible to a viewer than coherent noise (often called *aliasing*). Furthermore, in animation, we know that artifacts that flash on and off are much more annoying visually than a consistent error one way or the other, even if the numerical error is higher with the latter approach. Errors are not all the same. This topic could alone fill several theses (and is in the process of doing so).

Having an awareness of hardware. The Occam's Razor of computer graphics is that when an algorithm can be implemented on a system such that it lies on a better point of the cost/performance curve, then it is the preferred algorithm. This does not mean to suggest that we have a good definition for either *cost* or *performance*, but we have various indicative scalar measures for these multidimensional qualities. We differ from other areas of applied computer science only in that the architectural sweet spot for our graphics applications can change dramatically and dynamically. Furthermore, our need for both interactive performance and offline performance adds extra wrinkles to algorithm development. It is not easy and somewhat dangerous for algorithm development to follow too closely the fashions of hardware, but it is important to be aware of how algorithms in general will map to broad classes of hardware. We need to remain attuned to the trajectory in which hardware is going.

The idea that the development of algorithms can be divorced from their implementation is an acceptable simplification and abstraction for beginning undergraduates in computer science. It is inappropriate and downright wrong to cling to this notion when developing real graphics software. This comes from a fervent computer scientist!

Hardware can greatly affect the choice of graphics algorithm. The z-buffer algorithm, as we discussed earlier, completely dominates other visibility algorithms for this reason. Even if we did not have specialised computer graphics hardware to tip the scales, the design of microprocessors also has a great effect on our choices. For example, it will probably always be the case that algorithms having data structures that can be contiguously laid out in memory with a minimum of pointers will have far superior memory cache performance than heavily linked structures with little memory locality.

Using heuristics and experience. The demands of high-performance graphics systems requires every trick in the book. Need I say more?

4 Synergies

All this said, there are many problems in which synergies between computer graphics and computational geometry can be realised. I will not give references to specific results, and instead encourage potentially interested researchers to leaf through recent proceedings of *SIGGRAPH*, *Eurographics*, and *Graphics Interface*, while also looking at the relevant computer graphics journals. In the remaining space, I will outline some problems for which two heads may be better than one. Most of these problems are already well defined problems in computer graphics.

Parameterisation of irregular polygonal meshes. Although curved surfaces are the most significant modelling primitives in design applications, polygonal meshes have begun to dominate in entertainment applications (e.g., games), terrain models, and real-world data extracted by scanning instruments. Unlike traditional, regular polyhedral models, many new models are composed of meshes of polygons (typically triangles) of highly nonuniform density, aspect ratio, and area. The topology of these meshes is often uncertain *a priori*. This makes the two-dimensional *parameterisation* of such surfaces extremely difficult. Such a parameterisation is essential to the process of mapping two-dimensional textures onto these surfaces. A parameterisation of a mesh of polygons given by a set of vertices $\{V_i \in \mathbf{R}^3\}$ assigns a pair of values $U_i = (u_i, v_i)$ to each V_i. A large set of rules can be applied to the choice of these U_i, such as topological consistency, shape preservation, and so forth. Ultimately, however, most such parameterisations involve some kind of optimisation process in which the three-dimensional points V_i are "flattened" onto the plane with U_i being the projection of V_i on that plane. Unfortunately, very few surfaces can be flattened exactly, which pushes solutions of these problems toward an interesting blend of optimisation, discrete and continuous geometry, and topology.

Multiresolution algorithms and data representations. An increasingly common response to the endless need for increased data complexity is the use of data structures and algorithms that operate on, or construct, various levels of detail of data. The scale at which an algorithm operates at any given time is matched to factors such as the desired visual detail, machine speeds and bandwidths, data distribution, and so on. This opens up several interesting problems in geometry, including the need for new data structures, intersection and collision resolution, parameterisation, and surface tessellation, interrogation, manipulation and modification.

Query problems for subdivision surfaces. *Subdivision surfaces* are an important class of new modelling primitives that has recently arisen in computer graphics. In this case, an initial mesh of polygons is repeatedly (and possibly selectively) subdivided according to a set of rules. Each iteration of these rules provides a refinement of the initial mesh, thus giving an inherently multiresolution representation. Unlike traditional surfaces which are written in terms of smooth, generally polynomial, basis functions, an analytic form for the limit surface may not be available. In addition to those already mentioned above, there are some additional problems associated with subdivision surfaces, such as computing or approximating areas, tangents, trims, and intersections of parts of subdivision surfaces.

Mesh simplification. Because polygonal meshes have become so prevalent in computer graphics, we must also deal with the case of overly complex meshes. Reducing the number of polygons in dense meshes is an interesting problem that requires a significant amount of applied geometry so as to best preserve qualities such as curvature, tangency, area, etc. in the reduced model.

Incremental visibility. There is an increasing demand for making dynamic changes to complex geometric models. Given the size of many geometric models, even making a pass through all the data may impede interactive performance. New algorithms are required to effectively isolate changes in the visibility graph. One hopes that through a coherence assumption, the size of the changes is substantially smaller than the overall model.

Visibility and shadow computation for curved surfaces. Despite the trend toward polygonal models, many models still include algebraic and parametric surfaces. There are very few visibility and shadow computation algorithms operating directly on curved surfaces. Note that quadric shadow boundaries can arise even in polygonal models. The shadow boundaries for higher-order geometric models and light sources will be mathematically fearsome.

Silhouette computation and adaptive tessellation. It is easy to tell that much of computer graphics software is Canadian, because many of us refer to a characteristic problem of seeing the polygonal approximation to curved regions as "nickeling". By looking at the tessellations of curved geometric models into polygons, it is very clear that the place at which faceting is most evident lies at the *silhouette* of an object. This is an interesting geometric construction, since it involves computing the outline of a projected geometric surface. Silhouette computation has been an active area of research both in computer vision and computer graphics, and yet substantial progress in this area for the purposes of graphical rendering has not been forthcoming.

5 Conclusion

There is a somewhat happy ending to the Watkins' story. *Plane sweep* techniques, essentially two-dimensional generalisations of Watkins' algorithm, are quite prevalent today in the implementations of new illumination algorithms. Rendering systems of the future may well be the offspring of a marriage between z-buffer and Watkins'-style algorithms.

Computer graphics and computational geometry are areas that have a lot to share, but finding the right match between our areas of expertise and our cultures is less obvious that it at first seems. Computer graphics is an extremely pragmatic area, filled with a concern for the final image, and less concerned with the elegance of the complexity structure of geometric algorithms. Not all computational geometers will find our methods appealing, but I hope I have convinced some of you that it might be worth the effort to walk with us and get to know us better.

Acknowledgements/Disclaimer

The history reported in this paper is folkloric, and while it has been the subject of several conversations with some computer graphics "pioneers", I accept all blame for inaccuracies.

Reconstruction of 3-D Surface Object from its Pieces

GÖKTÜRK ÜÇOLUK İ. HAKKI TOROSLU

Dept. of Computer Engineering
Middle East Technical University, Ankara

ucoluk@ceng.metu.edu.tr
toroslu@ceng.metu.edu.tr

Abstract

The problem of reconstruction of broken surface objects embedded in 3-D space is handled. A coordinate independent representation for the crack curves is developed. A new robust matching algorithm is proposed which serves for finding matching pieces even when some brittle pieces are missing.

1 Introduction

The handled problem appears heavily in field archeology where reconstruction of hollow objects becomes a tedious and laborious task. It is the problem of jigsaw puzzle assembling of 3-D surfaces with no texture or color hints provided.

Previous work of [1, 2, 3] and the work of Wolfson [4] attack the 2-D problem and propose appropriate matching algorithms. Although Wolfson's algorithm is not the most efficient ($O(n \log n + \epsilon n)$) it is especially well designed to deal with noise. In his work, 2-D objects are represented by *shape signatures* that are strings which are obtained by polygonal approximation of the boundary curve. Freeman [5] describes 2-D shapes by a set of *critical points* (like discontinuities in curvature) and computes features between consecutive critical points. This method is weak in treating curves that do not possess such points. Ayache and Faugeras [6] attack a more difficult problem where rotation, translation and scale change is allowed. Their matching algorithm is based on finding correspondence between sides of polygons that approximate the 2-D shape curves. Another special feature based recognition technique is the one developed by Kevin *et al.*. This technique makes use of *breakpoints* and carry by nature the handicap mentioned for [5].

Works dealing with 3-D also exists. Kishon and Wolfson [7] introduce the arclength, curvature and torsion as signatures of a 3-D curve but decide not use torsion because its requirement to the third derivative. The matching problem is attacked as a longest substring search problem in their work. Kishon, in his work [8] proposes a spline fit which enables the easy incorporation of torsion as a stable signature. In [9], Schwartz and Sharir propose various metrics (like color on the boundaries) and a smoothing operation on the data.

There exists real world problems where a 2-D solution is insufficient *(Reconstruction from broken pieces of solid objects is one of them)* so a 3-D solid model is inevitable. Furthermore, in many of those real world problems a perfect match between two subjects is not possible. Environmental aging effects, imperfections in the digitization environment, the accumulation of systematic errors in numerical operations all contribute to this imperfection. Therefore, a robust, fault tolerant partial matching is required. This work proposes such a solution. In our work 3-D surface piece objects are represented by their boundary curves. These curves are parameterized by their *curvature* and *torsion* scalars which are calculated from the discrete 3-D boundary data and quadratically added to form a circular string of a single value. A noise tolerant matching algorithm serves to find the best match of two such circular strings even for cases where the match is fragmented.

2 Mathematical Representation of the Problem

We will assume that the object which will be reassembled has no thickness, hence can be represented by a surface in a 3-D Euclidean space. The pieces of a surface structure embedded in a 3-D space are surfaces with boundaries that are closed curves of the 3-D space. Since a matching over these closed curves corresponds to the task of reassembling, a coordinate independent parameterization of these curves are very desirable. The fundamental theorem of the local theory of curves (see [10, 11]) reads as

> *Given differentiable functions $\kappa(s) > 0$ and $\tau(s)$, $s \in \mathbf{I}$, there exists a regular parameterized curve $\vec{r} : \mathbf{I} \to \mathbf{R}^3$ such that s is the arc length, $\kappa(s)$ is the curvature, and $\tau(s)$ is the torsion of $\vec{r}$.*

Moreover, any other curve $\vec{r}^*$, satisfying the same conditions, differs from $\vec{r}$ by a rigid motion; that is, there exists an orthogonal linear map Ω of $\mathbf{R}^3$, with positive determinant, and a vector $\vec{c}$ such that $\vec{r}^* = \Omega \circ \vec{r} + \vec{c}$.

What we can conclude from this theorem is exactly what we were looking for:

If two different curves which are parameterized by their arc length produce the same torsion and curvature values then we can conclude that these curves are the same (modulo rotation and translation).

Furthermore, the converse is also true. *Curvature* is defined as

$$\kappa = |\vec{r}''|$$

Torsion is defined as

$$\tau = \frac{1}{\kappa^2}[\vec{r}'\vec{r}''\vec{r}''']$$

where the square brackets $[\cdots]$ have the special meaning of

$$\left[\vec{A}\vec{B}\vec{C}\right] \equiv \begin{vmatrix} A_x & A_y & A_z \\ B_x & B_y & B_z \\ C_x & C_y & C_z \end{vmatrix}$$

Furthermore the prime denotes differentiation with respect to the arc length s:

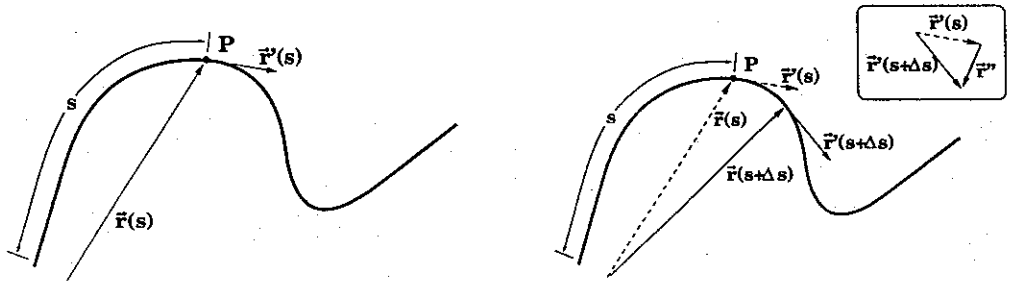
$$\vec{r}' = \frac{d\vec{r}}{ds}$$

As known s is defined by:

$$s(t) = \int_0^t ds = \int_0^t \sqrt{d\vec{r}\cdot d\vec{r}} = \int_0^t \sqrt{dx^2 + dy^2 + dz^2}$$

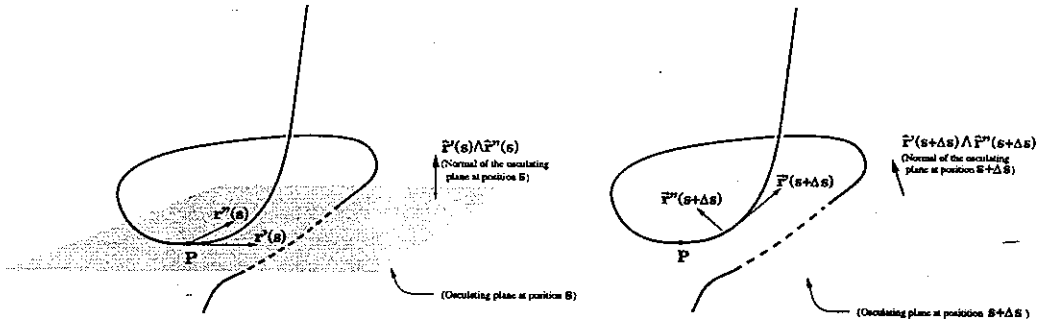
Where t is the *parameter* of the curve that maps each value in an interval in $\mathbf{R}$ into a point $r(t) = (x(t), y(t), z(t)) \in \mathbf{R}^3$ in such a way that the functions $x(t)$, $y(t)$, $z(t)$ are differentiable.

Intuitively speaking, the *curvature* at a point on the curve is the measure of how rapidly the curve pulls away from the tangent line at that point (so in a close neighborhood of that point we will have a deviated tangent line).



Tangent is nothing else than the change in the position vector $\vec{r}$ namely $\vec{r}'$. The magnitude of the change rate of this vector $|\vec{r}''|$is called *curvature*.

Consider at any point on the curve the plane formed to include the vectors $\vec{r}'$ and $\vec{r}''$ (at that point). This plane is called the osculating plane of that point. Again intuitively speaking, the *torsion* at a point on the curve is the measure of how rapidly the curve pulls away form the osculating plate at that point (so in a close neighborhood of that point we will have a deviated osculating plane).



Osculating plane is the plane that contains the $\vec{r}'$ and $\vec{r}''$ vectors. Of course this plane changes from point to point. *torsion* is the scalar measure of the rate of deviation of this plane (the deviation of the normal of the plane). *torsion* is defined as the change in the magnitude of this deviation. This is so because calculation reveals that the direction of the change is always in the direction of $\vec{r}''$

In the discrete case we have instances of r which are labeled with an index i. We assume that the labeling is done such that for any two r_i and r_{i+1} instances there exist no provided r_k value that corresponds to a curve point that is between them. Hence, the index is the discrete form of the *curve parameter*. Differentials will be replaced by differences with the following definitions

$$\Delta x_i = x_i - x_{i-1} \qquad \Delta y_i = y_i - y_{i-1} \qquad \Delta y_i = y_i - y_{i-1}$$

$$\Delta s_i = \sqrt{\Delta x_i^2 + \Delta y_i^2 + \Delta z_i^2}$$

So for the arc length we have $s_i = \sum_{k=1}^{i} \Delta s_i$. Once obtained the tuples $(\vec{r}_i, s_i)$ the $\vec{r}', \vec{r}'', \vec{r}'''$ are calculated for equally spaced (δs) points in the usual manner. To avoid local divergent behaviors the derivatives are calculated as an *average* value in a given radius of neighborhood. Experimentation has shown that a δs value which is large enough to accommodate ~ 20 Δs_i values performs very well.

The κ_i and τ_i values form a 2-dimensional feature vector ξ_i. The sequence of feature vectors ξ_i forms the shape signature string. Since the objects dealt with are defined to have closed boundary curves, in all algorithms operating on the shape signature strings the assumption that these strings round over (i.e. be circular) will be made.

3 The Matching Algorithm

The broken pieces might have worn off contours. Therefore matching the algorithm shall be

- robust in matching (i.e. fault tolerant),
- allow the non-existence of some minor pieces.

In Figure 1 two pieces with some missing portion and the affect of this on the string representation is illustrated. In the chosen representation, this corresponds to

- accepting numerical matches with an ε tolerance,
- being able to resume the match after a gap of non-matching data.

The devised algorithm to match two curves represented respectively by the strings $\xi_i|_1^R$ and $\eta_i|_1^C$ (ξ_i and η_i are feature vectors) is as follows: we define a matrix Λ as $\Lambda_{ij} = \| \xi_i - \eta_j \|$. So Λ is a symmetric with nonnegative entries.

In the following algorithm a two dimensional array M is filled out. M will be holding the start and end positions of the matching segments. So, one index takes values as *start* or *end*. The second index runs through an enumeration of the found matching segments. M_p^{start} and M_p^{end} hold the start and end *position informations* of the found p^{th} segment, respectively. A position information of a start (or end) is a pair of indices, namely the row and column numbers of the Λ matrix where the segment starts (or ends).

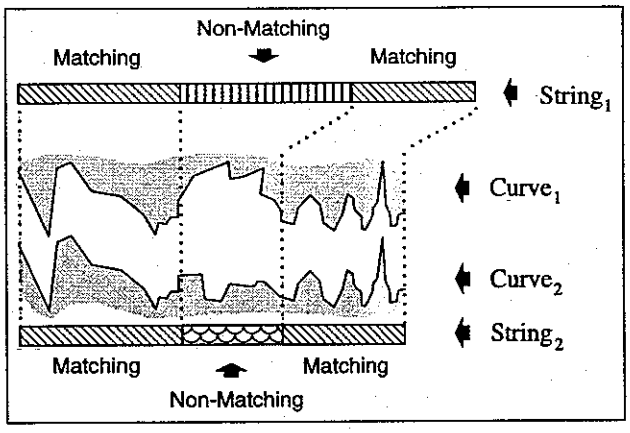


Figure 1: Two matching segments having a missing part

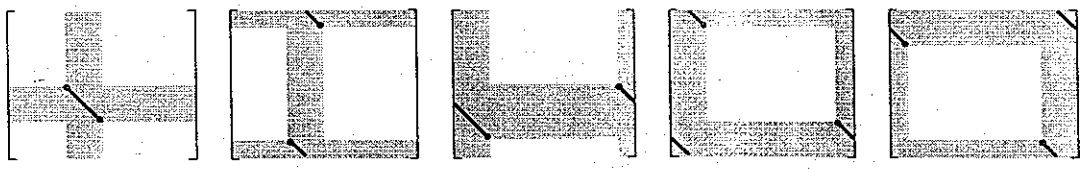
$match() \leftarrow \{\ S \leftarrow \min\{R, C\}$
$p \leftarrow 0$
for $i \leftarrow 1 \ldots R$ **do**
for $j \leftarrow 1 \ldots C$ **do**
if $\Lambda_{ij} \wedge \Lambda_{predecessor(i,j)} > \varepsilon$ **then**
$\{\ (k,l) \leftarrow (i,j)$
$m \leftarrow 0$
repeat $\{\ m \leftarrow m + 1$
$(k,l) \leftarrow successor(k,l)\ \}$
until $m \geq S \vee \Lambda_{kl} > \varepsilon$
$p \leftarrow p + 1$
$\mathbf{M}_p^{start} \leftarrow (i,j)$
$\mathbf{M}_p^{end} \leftarrow predecessor(k,l)\ \}\ \}\ \}$

$predecessor(i,j) \leftarrow \{$ **if** $i = 1$ **then** $k \leftarrow R$ **else** $k \leftarrow i - 1$
if $j = 1$ **then** $l \leftarrow C$ **else** $l \leftarrow j - 1$
return $(k,l)\}\ \}$

$successor(i,j) \leftarrow ((i \bmod R) + 1, (j \bmod C) + 1)$

From know on, denotationally, we will represent segments by a naming (e.g. α, β or α_i). Each segment, naturally, has four values associated: its start position (a *row* and a *column* number) in the matrix Λ and its end position (a *row* and a *column* number). These are represented by the appropriate combination of an superscript which is either *start* or *end* and a subscript that is either *row* or *column*.

The next task is to determine, among the segments found, which can follow which. As was stated, due to the circular structure of the matched curves a special treatment is necessary in finding the answer to this question. To avoid the halting problem of the algorithm we impose a canonical order onto the concept of following. The canonical order we will impose says that if a segment β is *following* a segment α then $\alpha_{row}^{end} < \beta_{row}^{start}$. Of course this is 'a necessary but not sufficient' criteria that has to be met. (The converse is not always true: you can have *non-following* two segments α and β where $\alpha_{row}^{end} < \beta_{row}^{start}$ still holds). To complete the definition of the following segments we consider the possible positions of a segment α (which is going to be followed by β) in the Λ matrix *(light shaded areas are forbidden zones for the following segment to start in due to the imposed canonical order but it may end in there; dark shades are the regions where an overlapping would occur, so the following segment shall have no points in there).*



CASE:I CASE:II CASE:III CASE:IV CASE:V

Start and end point-wise, CASE IV and CASE V are not different from each other, so they will be considered as the same. We define a comparison operator $\prec$ that will admit two segments as operands and return True if the right operand is a *following segment* of the left one and False otherwise. Formally this operator can be defined as *(we are making use of the mathematical notation for representing closed/open/semiclosed sets; in our cases set elements are integral values)*:

$$\begin{aligned}
\alpha \prec \beta \quad \leftarrow \quad & \textbf{if } wrapped_{row}(\alpha) \textbf{ then } \beta_{row}^{start} \in (\alpha_{row}^{end}, \alpha_{row}^{start}) \ \wedge \ \beta_{row}^{end} \in (\beta_{row}^{start}, \alpha_{row}^{start}) \\
& \qquad\qquad \textbf{else } \beta_{row}^{start} \in (\alpha_{row}^{end}, R] \ \wedge \ \beta_{row}^{end} \in (\beta_{row}^{start}, R] \cup [1, \alpha_{row}^{start}) \\
& \wedge \\
& \textbf{if } wrapped_{col}(\alpha) \textbf{ then } \beta_{col}^{start} \in (\alpha_{col}^{end}, \alpha_{col}^{start}) \ \wedge \ \beta_{col}^{end} \in (\beta_{col}^{start}, \alpha_{col}^{start}) \\
& \qquad\qquad \textbf{else } \beta_{col}^{start} \in [1, \alpha_{col}^{start}) \cup (\alpha_{col}^{end}, C] \\
& \qquad\qquad\qquad \wedge \\
& \qquad\qquad\qquad \textbf{if } \beta_{col}^{start} \in (\alpha_{col}^{end}, C] \textbf{ then } \beta_{col}^{end} \in [1, \alpha_{col}^{start}) \cup (\alpha_{col}^{end}, C] \\
& \qquad\qquad\qquad\qquad\qquad \textbf{else } \beta_{col}^{end} \in [1, \alpha_{col}^{start})
\end{aligned}$$

Where we have defined $wrapped_{\delta}(\chi) \ \leftarrow \ \chi_{\delta}^{start} > \chi_{\delta}^{end} \quad , \quad \delta \in \{row, col\}$

The $\prec$ operator will yield always the correct answer for the cases where α is following β or visa versa. For segments, though, that have overlapping regions the answer is undetermined. Hence we are able to define a partial order among the set of all found segments, namely the M array. So, we will perform a topological sort on the M array where $\prec$ is the ordering criteria. A simple bubble sort will do the job *($\leftrightarrow$ stands for 'content exchange')*:

$$\begin{aligned}
& sort_M() \leftarrow \textbf{for } j \leftarrow 1 \ldots lastindex(\mathrm{M}) \textbf{ do} \\
& \qquad \textbf{for } i \leftarrow 1 \ldots (lastindex(\mathrm{M}) - j) \textbf{ do} \\
& \qquad\quad \textbf{if } \mathrm{M}_{i+1} \prec \mathrm{M}_i \textbf{ then } \mathrm{M}_i \leftrightarrow \mathrm{M}_{i+1}
\end{aligned}$$

The remaining of the algorithm is basically a search for the longest path in a graph where vertices are the segments and unidirectional edges between these vertices are introduced from segments to the *following* segments. The weights on the vertices are nothing else than the lengths of the segments. The task of finding the longest match is converted into a task in which the longest path of the described graph is found. The longest path will yield a maximal weight (length of segments) sum of the vertices (segments) on the path. To start a search for the longest path we have to identify the *starting* and *terminating* vertices of the graph (i.e. those segments which are not following any segment and segments which are not followed by any segment). By two linear scans over the M array we are able to identify and mark those segments:

$$\begin{aligned}
& mark_terminatings() \leftarrow \{ \textbf{for } i \leftarrow 1 \ldots lastindex(\mathrm{M}) \textbf{ do } mark_as_terminating(\mathrm{M}_i) \\
& \qquad r \leftarrow [\mathrm{M}_{lastindex(\mathrm{M})}]_{row}^{start} \\
& \qquad c \leftarrow [\mathrm{M}_{lastindex(\mathrm{M})}]_{col}^{start} \\
& \qquad \textbf{for } i \leftarrow (lastindex(\mathrm{M}) - 1) \ldots 1 \textbf{ do} \\
& \qquad\quad \textbf{if } [\mathrm{M}_i]_{row}^{end} < r \ \wedge \ [\mathrm{M}_i]_{col}^{end} < c \textbf{ then } remove_terminating_mark(\mathrm{M}_i) \\
& \qquad\qquad\qquad\qquad \textbf{else if } [\mathrm{M}_i]_{row}^{start} > r \ \wedge \ [\mathrm{M}_i]_{col}^{start} > c \textbf{ then} \\
& \qquad\qquad\qquad\qquad\quad \{ r \leftarrow [\mathrm{M}_i]_{row}^{start} \\
& \qquad\qquad\qquad\qquad\quad\ \ c \leftarrow [\mathrm{M}_i]_{col}^{start} \ \} \ \}
\end{aligned}$$

$$\begin{aligned}
& mark_startings() \leftarrow \{ \textbf{for } i \leftarrow 1 \ldots lastindex(\mathrm{M}) \textbf{ do } mark_as_starting(\mathrm{M}_i) \\
& \qquad r \leftarrow [\mathrm{M}_1]_{row}^{end} \\
& \qquad c \leftarrow [\mathrm{M}_1]_{col}^{end} \\
& \qquad \textbf{for } i \leftarrow 2 \ldots lastindex(\mathrm{M}) \textbf{ do} \\
& \qquad\quad \textbf{if } [\mathrm{M}_i]_{row}^{start} > r \ \wedge \ [\mathrm{M}_i]_{col}^{start} > c \textbf{ then } remove_starting_mark(\mathrm{M}_i) \\
& \qquad\qquad\qquad\qquad \textbf{else if } [\mathrm{M}_i]_{row}^{end} < r \ \wedge \ [\mathrm{M}_i]_{col}^{end} < c \textbf{ then} \\
& \qquad\qquad\qquad\qquad\quad \{ r \leftarrow [\mathrm{M}_i]_{row}^{end} \\
& \qquad\qquad\qquad\qquad\quad\ \ c \leftarrow [\mathrm{M}_i]_{col}^{end} \ \} \ \}
\end{aligned}$$

For convenience we introduce two dummy vertices, namely at start M_0 and at end M_λ (where $\lambda = lastindex(\mathrm{M}) + 1$) with length zero and satisfying the conditions

$$\begin{aligned}
\forall i \ni marked_as_terminating(\mathrm{M}_i) \ &: \ \mathrm{M}_0 \prec \mathrm{M}_i \\
\forall i \ni marked_as_starting(\mathrm{M}_i) \ &: \ \mathrm{M}_i \prec \mathrm{M}_\lambda
\end{aligned}$$

The following greedy algorithm uses dynamic programming to find the longest path. Gradually it fills out an array that we will name as *longest*:

$$find\_longest\_path() \leftarrow \textbf{for } i \leftarrow lastindex(\mathrm{M}) \ldots 0 \textbf{ do}$$
$$longest_i \leftarrow \max\{\, \omega(length(\mathrm{M}_k)) + longest_j \mid \mathrm{M}_i \prec \mathrm{M}_j \,\}$$

length is a function returning the count of match points of the segment given as argument to it:

$$length(\alpha) \leftarrow \textbf{if } \alpha_{row}^{start} < \alpha_{row}^{end} \textbf{ then } \alpha_{row}^{end} - \alpha_{row}^{start} \textbf{ else } R + \alpha_{row}^{end} - \alpha_{row}^{start}$$

$\omega()$ is a function which defines the weight contribution of the count of match points for a continuous match segment given to it as the argument. The idea is to allow a penalty treatment for short matches. If no such penalty is favored then it is possible to simply define $\omega(m) = m$.

4 Conclusion

We presented a method for matching two closed space curves which are holding discrete feature values, in a robust manner. Unlike in other related works the problem of the proper treatment of missing parts in a match is put under focus and a complete solution is proposed. The reconstruction of the object is just an exhaustive search over all 'pieces' and choosing the best fittings. The idea is simple:

- Find the best match.
- Join the matching portions (perform in parallel the necessary bookkeeping).
- Removing the parts of the joint obtain the representation of a single piece.
- Add this new obtained piece and remove the two pieces which were joined from the database, hence reducing the count of pieces by one, continue until only one piece is left.

For a possible implementation we would propose a visual workbench approach in which the user has a full control over the matching parameters and the matching itself and the availability of an undo operation over the construction history. A project of such an implementation has been started.

Further efforts can go into the implementation details where a suitable data representation and efficient retrieval mechanisms will be the main concern.

References

[1] H. Freeman and L. Garder. A pictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. *IEEE Trans. Electron. Comput.*, EC-13:118–127, 1964.

[2] G. M. Radack and N. I. Badler. Jigsaw puzzle matching using a boundary-centered polar encoding. *Comput. Graphics Image Processing*, 19:1–17, 1982.

[3] H. Wolfson, E. Schonberg, A Kalvin, and Y. Lambdan. Solving jigsaw puzzle using computer vision. *Ann. Oper. Res.*, 12:51–64, 1988.

[4] H. Wolfson. On curve matching. *IEEE, Trans. Pattern. Anal. Machine. Intell.*, 12:483–489, 1990.

[5] H. Freeman. Shape description via the use of critical points. *Pattern Recogn.*, 10:159–166, 1978.

[6] N. Ayache and O. D. Faugeras. Hyper: A new approach for the recognition and positioning of two-dimensional objects. *IEEE, Trans. Pattern. Anal. Machine. Intell.*, 8:44–54, 1986.

[7] E. Kishon and H. Wolfson. 3-d curve matching. In *Proceeding of the AAAI Workshop on Spatial Reasoning and Multi-sensor Fusion*, pages 250–261, 1987.

[8] E. Kishon, T. Hastie, and H. Wolfson. 3d curve matching using splines. In *First Europian Conference on Computer Vision*, pages 589–591, 1990.

[9] J. T. Schwartz and M. Sharir. Identification of partially obscured objects in two and three dimension by matching noisy characteristic curves. *IEEE, Trans. Pattern. Anal. Machine. Intell.*, 8:44–54, 1986.

[10] M. P. do Carmo. *Differential geometry of curve and surfaces.* Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[11] A. Goetz. *Introduction to differential geometry of curve and surfaces.* Prentice-Hall, Englewood Cliffs, New Jersey, 1970.

Sampling and Reconstructing Manifolds Using Alpha-Shapes

Fausto Bernardini* Chandrajit L. Bajaj†

Department of Computer Sciences
Purdue University‡

Abstract

There is a growing interest for the problem of reconstructing the shape of an object from multiple range images. Several methods, based on heuristics, have been described in the literature. We propose the use of alpha-shapes, which allow us to give a formal characterization of the reconstruction problem and to prove that, when certain sampling requirements are satisfied, the reconstructed alpha-shape is homeomorphic to the original object and approximate it within a fixed error bound.

In this paper, we give a formal proof of sampling requirements for the reconstruction of 1-manifolds in $\mathbf{R}^2$, and briefly sketch practical applications of alpha-shapes to the reconstruction of three-dimensional CAD models from digital scans.

1 Introduction

Cheaper, easier-to-use 3D digitizers are fostering a growing interest for the problem of *shape-reconstruction*. Automatic methods for reconstructing an accurate geometric model of an object from a set of digital scans have applications in reverse engineering, shape analysis, virtual worlds authoring, 3D faxing and tailor-fit modeling.

Range or optical-triangulation laser scanners produce a regular grid of measurements, which can be easily converted to a rectangular or cylindrical surface model when a single scan suffices to capture the whole object's surface. However more often multiple scans are required, and the results must be merged together. Several approaches have been proposed to reconstruct the shape of an object from a collection of digital scans.

Turk and Levoy [11] proposed to "zipper" together several meshes obtained from separate 3D-scans of an object. More recently, Curless and Levoy [5] presented an approach to merge several range images by scan-converting each image to a weighted signed distance function in a regular 3D grid. The zero-contour of the signed distance function, which can be easily extracted with a marching cubes algorithm [10], represents the reconstructed surface.

A different class of methods try to rely on spatial location of points only, without any assumed knowledge of connectivity between sampled points. Boissonnat [3] proposes two methods to build a triangulation having the given points as vertices. Following his first approach, one starts with creating an edge between the two closest points. A third point is then chosen and added, so that a triangle is formed. Other points are successively added and new triangles are created, and joined to an edge of the current triangulation boundary, until all points have been included. The second method is based on the idea of first computing a Delaunay triangulation of the convex hull of the set of points, and then *sculpturing* the volume by removing tetrahedra, until all points are on its boundary, or no tetrahedra can be further removed.

Choi *et al.* [4], described a method to incrementally form a triangulation interpolating all data points, based on the assumption that there exists a point from which all the surface is visible. After a triangulation is built, it is improved by edge swapping based on a smoothness criterion.

Veltkamp [12] introduced a new general geometric structure, called γ-graph. The γ-graph coincides initially with the convex hull of the data points, and is progressively *constricted* (i.e. tetrahedra having boundary faces are deleted) until the boundary of the γ-graph is a closed surface, passing through all the given points.

Hoppe *et al.* [9] compute a signed distance function from the data points, and then use its zero-contour as

*Current address: IBM T. J. Watson Research Center
P.O. Box 704 Yorktown Heights, NY 10538
fausto@watson.ibm.com

†bajaj@cs.purdue.edu

‡West Lafayette, Indiana
47907-1398 USA

http://www.cs.purdue.edu/research/shastra

an approximation of the object. To define the signed distance from the unknown surface, they compute a best-fit tangent plane for each data point, and then find a coherent orientation for the surface by propagating the normal direction from point to point, using a precomputed minimum spanning tree to favor propagation across points whose associated normals are nearly parallel.

One of the most difficult problems of shape reconstruction from unorganized points is understanding how to "connect-the-dots" so as to form a surface that has the same topological (e.g. number of handles) and geometric (e.g. depressions and protrusions) characteristics of the original. All the methods listed above are based on geometric heuristics. While these methods have been shown to be successful on several examples and practical applications, they fail to provide requirements on the sampling that guarantee a provably correct reconstruction.

Alpha-shapes were introduced in the plane by Edelsbrunner *et al.* in [7] and then extended to higher dimensions [6, 8], as a geometric tool for reasoning about the "shape" of an unorganized set of points. They offer the dual benefit of having a solid mathematical foundation and of being relatively easy to compute. We have developed several automatic reconstruction methods based on alpha-shapes and algebraic-patch fitting [1, 2].

In this paper we formalize the shape reconstruction problem, give a set of sufficient conditions for reconstructing an object using alpha-shapes, and discuss some practical considerations.

2 Statement of the Problem

Reconstructing the shape of an object from an unorganized "cloud" of points is in general an underconstrained problem. Consider the simple 2D reconstruction problem illustrated in Figure 1: Several solutions are possible, and it is difficult to identify the "best" among them. It is therefore of interest looking at the following problem: What are the characteristics of a sampling S (a finite set of points) of the surface of a solid object M, such that M can be reconstructed from S unambiguously and within predefined approximation bounds?

In particular, we consider the following *reconstruction problem*: Starting with a sampling of the surface B of a solid, we want to compute a triangulated surface K that has the "same shape" of B, and such that a suitably defined *distance* $D(K, B)$ of K from B is bounded by a given ε. A useful distance measure is for example:

$$D(K,B) = \max_{p\in[K]} \min_{q\in B} ||p-q||.$$

Stated formally:

Problem 2.1 *Let B be a compact 2-manifold without boundary (in particular, the boundary of a solid M), and $S \subset B$ a finite set of points (sampling). Construct a (geometric) simplicial complex K, such that $K^{(0)} = S$, K is homeomorphic to B, and $D(K,B) < \varepsilon$, for a fixed $\varepsilon \in \mathbf{R}, \varepsilon > 0$.*

The pair (K, h), where K is a simplicial complex and h is a homeomorphism $h : [K] \to B$ is called a *triangulation* in algebraic topology[1]

An algorithm aimed at reconstructing the shape of an object from point data alone must have a way of inferring spatial relationships among points. Characteristics of the sampling that guarantee an unambiguous and correct reconstruction depend on how the data is interpreted by the algorithm.

We have already mentioned that alpha-shapes allow us to find spatial relationships between points of an unorganized set. The relationships are based on proximity. Clusters of points close to each other are grouped to form edges, triangles and tetrahedra, and more complex structures made of collections of these simple constituents.

The question we need to answer is therefore the following: What are sufficient conditions of a sampling that guarantee that there exists an α such that the corresponding α-shape satisfies the requirements of Problem 2.1?

3 Sampling Requirements

We can look at the two-dimensional case to get some insight into the problem. Figure 2 illustrates the discussion that follows. In this case, we are sampling a 1-manifold B (observe that B is a collection of "loops"). Intuitively, we can think of the points of the sampling as "pins" that we fix on B. We now use a disk probe of radius $\rho = \sqrt{\alpha}$ to "sense" the manifold. The probe must be able to move from point to point of S on the surface, touching pairs of points in sequence, and without touching other parts of B. The pairs of points will be connected by segments of the alpha-shape, and will form loops homeomorphic (and geometrically close) to each component of B.

Clearly, a necessary condition is that no two adjacent points of the sampling are farther away than the diameter of our disk-probe, because otherwise the probe would "fall" inside the boundary of our solid object. We also need to make sure that all, and only, the edges

[1] For notation see the full version of the paper, available on the World Wide Web page for CCCG97. The notation used for alpha-shapes follows that used in reference [6].

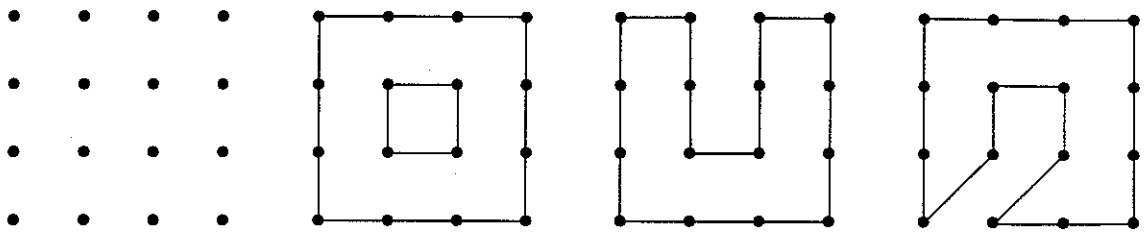


FIGURE 1: An example of ambiguous 2D reconstruction from points. From left to right: A point sampling and three, equally acceptable, reconstructions.

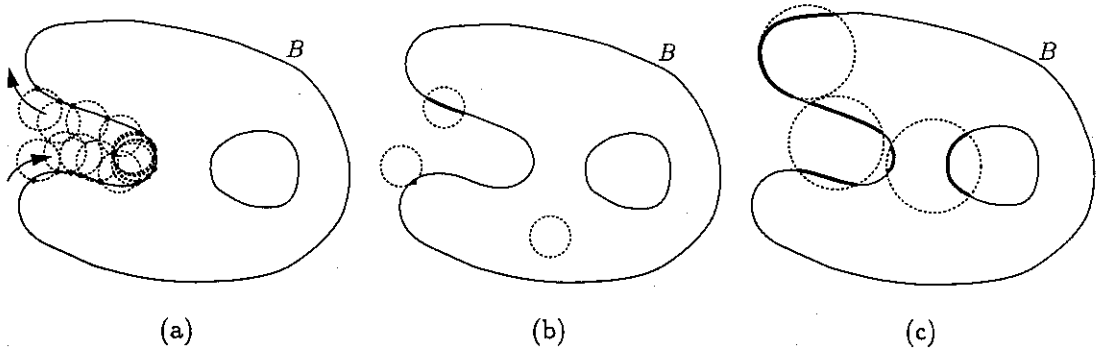


FIGURE 2: Sampling requirements for 1-manifolds in $\mathbf{R}^2$. (a) The sampling density must be such that the center of the "disk probe" is not allowed to cross B without touching a sample point. (b) The radius ρ of the disk probe must be small enough that the intersection with B has at most one connected component. (c) Examples of non admissible cases of probe-manifold intersections.

connecting pairs of adjacent points are α-exposed. To do this, our probe needs to be small enough to be able to isolate a neighborhood of a point p on B, or, equivalently, discern "adjacent" points on B from points that are close in the Euclidean sense but not on the surface. These requirements are formalized in the following

Theorem 3.1 *Let $B \subset \mathbf{R}^2$ be a compact 1-manifold without boundary, and $S \subset B$ a finite point set. If*

1. *For any closed disk $D_\rho \subset \mathbf{R}^2$ of radius ρ, $B \cap D_\rho$ is either (a) empty; (b) a single point p (then $p \in \mathrm{bd}(D_\rho)$); (c) homeomorphic to a closed 1-ball I, such that $\mathrm{int}(D_\rho) \cap B = \mathrm{int}(I)$;*
2. *An open disk of radius ρ centered on B contains at least one point of S,*

then the alpha-shape $\mathcal{S}_\alpha$ of S, $\alpha = \rho^2$, is homeomorphic to B and

$$D(\mathcal{S}_\alpha, B) = \max_{p \in \mathcal{S}_\alpha} \min_{q \in B} ||p - q|| < \rho.$$

Observe that B is in general a collection of 1-spheres B_i. We will prove the theorem by showing that for each 1-sphere $B_i \subset B$ there is a homeomorphic component in $\mathcal{S}_\alpha$, and then showing the bound on the distance.

Before we prove the theorem, we need a few lemmas. In the lemmas that follow, B, B_i, and S are those defined above. The symbol D_ρ is used as above to indicate a closed disk of radius ρ. We refer to the two conditions stated in the theorem as conditions 1 and 2. We often refer to two points p, q on a component B_i of B, and use the symbols X, Y to indicate the two closed 1-balls on the 1-sphere B_i having p, q as boundary points. Obviously $X \cup Y = B_i$.

Lemma 3.1 *Let p, q be two points on B. If there exists D_ρ such that $p, q \in \mathrm{bd}(D_\rho)$, then $D_\rho \cap B$ is a 1-ball I, and $\mathrm{bd}(I) = \{p, q\}$.*

Proof: Since D_ρ contains two points of B, by condition 1 it must intersect B in a (closed) 1-ball I, with $p, q \in I$. Suppose $p \notin \mathrm{bd}(I)$. Then $p \in \mathrm{int}(I)$. But $p \notin \mathrm{int}(D_\rho) \cap B$, therefore condition 1 cannot be satisfied. ◇

Lemma 3.2 *Let p, q be two points on B_i, and let X, Y be the two 1-balls on B_i, $\mathrm{bd}(X) = \mathrm{bd}(Y) = \{p, q\}$. If*

there exists D_ρ such that $p, q \in \text{bd}(D_\rho)$, then either $D_\rho \cap B = X$ or $D_\rho \cap B = Y$.

Proof: By Lemma 3.1, $D_\rho \cap B$ is a 1-ball whose boundary is $\{p, q\}$. Clearly this 1-ball must be a subset of B_i. There are only two 1-balls on B_i having $\{p, q\}$ as boundary, namely X and Y. ◇

Lemma 3.3 *Let p, q be two points on B_i, and let X, Y be the two 1-balls on B_i, $\text{bd}(X) = \text{bd}(Y) = \{p, q\}$. If $\text{int}(X) \cap S = \emptyset$ then $||p - q|| < 2\rho$.*

Proof: Suppose that $||p - q|| \geq 2\rho$. Since X is a 1-ball connecting p and q and $||p - q|| \geq 2\rho$, there exists a point $c \in X$ such that $||p - c|| = \rho$. Consider D_ρ centered in c, and observe that $p \in \text{bd}(D_\rho)$. Since D_ρ contains two points of B (p and c), it must intersect B in a 1-ball I, and p must be a boundary point of I, by condition 1.

The other boundary point of I must be contained in the 1-ball Z between c and q. Notice that q cannot be in $\text{int}(D_\rho)$ because $||p - q|| \geq 2\rho$. Also, there are no other points of S in $Z \subset \text{int}(X)$. Therefore $\text{int}(D_\rho)$ is an open disk of radius ρ centered on B that contains no points of S, contradicting condition 2. ◇

Lemma 3.4 *Let p, q be two points on B_i, and let X, Y be the two 1-balls on B_i, $\text{bd}(X) = \text{bd}(Y) = \{p, q\}$. If $\text{int}(X) \cap S = \emptyset$ then there exists D_ρ such that $p, q \in \text{bd}(D_\rho)$ and $D_\rho \cap B = X$.*

Proof: Notice that by Lemma 3.2, either $D_\rho \cap B = X$, or $D_\rho \cap B = Y$. It will therefore suffice to show that there must be a point of X other than p, q in D_ρ.

By Lemma 3.3, $||p - q|| < 2\rho$, and therefore there are two disks $D_{1,\rho}, D_{2,\rho}$ such that $p, q \in \text{bd}(D_{k,\rho}), k = 1, 2$, whose centers lie on the opposite sides of the line through p, q. Assume that there are no points of X other than p, q in either of these disks.

Consider the line through the midpoint of segment p, q and orthogonal to the segment. This line must intersect X at a point c, which lies outside the two disks. It is easy to see that $||c - p|| = ||c - q|| > \sqrt{2}\rho > \rho$. Then take the disk D_ρ centered in $c \in X$. Since it contains a point of B in its interior, it must intersect B in a 1-ball I containing c, by condition 1. Observe that I cannot include p or q because of the bound on the distance. Therefore, I must be a proper subset of X. Since X does not contain points of S in its interior, $\text{int}(D_\rho)$ violates condition 2. ◇

Lemma 3.5 *Let p, q be two points on B_i, and let X, Y be the two 1-balls on B_i, $\text{bd}(X) = \text{bd}(Y) = \{p, q\}$. If $\text{int}(X) \cap S = \emptyset$ then there exist two disks $D_{1,\rho}, D_{2,\rho}$ such that $p, q \in \text{bd}(D_{k,\rho}), k = 1, 2$ and $D_{1,\rho} \cap B = D_{2,\rho} \cap B = X$.*

Proof: Let the two disks $D_{k,\rho}$ be as in Lemma 3.4. By that same lemma, one of the two disks, say $D_{1,\rho}$ must be such that $D_{1,\rho} \cap B = X$. Then assume that for the other disk $D_{2,\rho} \cap B \neq X$. By Lemma 3.2 we must have $D_{2,\rho} \cap B = Y$. All of B_i is then contained in the union of the two disks.

Now consider a disk $D_\rho(t)$ centered in $c = tc_1 + (1-t)c_2$, where c_1, c_2 are the centers of $D_{1,\rho}$ and $D_{2,\rho}$, respectively. For $0 \leq t \leq 1$ the disk moves from a position coincident with $D_{1,\rho}$ to one coincident with $D_{2,\rho}$. For each $0 \leq t \leq 1$, $D_\rho(t)$ contains p and q, and therefore, to satisfy condition 1, must contain all X or all Y, but can never contain both.

For any point $x \in \text{int}(X)$ the function

$$f_x(t) = ||x - c(t)|| - \rho$$

is continuous, and negative for $t = 0$. Since $D_\rho(1) \cap \text{int}(X) \neq \text{int}(X)$, there exists $\overline{x} \in \text{int}(X)$ such that $f_{\overline{x}}(1) > 0$. Then there is a $0 < \overline{t} < 1$ such that $f_{\overline{x}}(\overline{t}) = 0$. Let $\overline{x}$ be the point for which is minimum the $\overline{t}$ that makes $f_{\overline{x}}(\overline{t})$ zero.

Then X lies all in $D_\rho(\overline{t})$, and $\overline{x}$ lies on the boundary of $D_\rho(\overline{t})$. Since $\overline{x} \in \text{int}(X)$, and $p, q \in \text{int}(D_\rho(\overline{t}))$, $I = D_\rho(\overline{t}) \cap B$ contains $\overline{x}$ in its interior. But then condition 1 cannot be satisfied. ◇

Lemma 3.6 *Consider two points $p, q \in S$. If $p \in B_i$ and $q \in B_j$, $i \neq j$, then the segment $\sigma_T, T = \{p, q\}$ is not α-exposed.*

Proof: For σ_T to be α-exposed there must exist a D_ρ such that $p, q \in \text{bd}(D_\rho)$. But then $D_\rho \cap B$ must be a 1-ball by condition 1, which is impossible since p, q belong to different components of B. ◇

Lemma 3.7 *Consider two points $p, q \in S$, with $p, q \in B_i$, and let X, Y be the two 1-balls on B_i, $\text{bd}(X) = \text{bd}(Y) = \{p, q\}$. If both $\text{int}(X)$ and $\text{int}(Y)$ contain points of S, then the segment $\sigma_T, T = \{p, q\}$ is not α-exposed.*

Proof: If there exists D_ρ such that $p, q \in \text{bd}(D_\rho)$, then by Lemma 3.2 D_ρ must contain either $\text{int}(X)$ or $\text{int}(Y)$. Since both contain points of S, σ_T cannot be α-exposed. If the disk D_ρ does not exists that σ_T cannot be α-exposed. ◇

Lemma 3.8 *Consider two points $p, q \in S$, with $p, q \in B_i$, and let X, Y be the two 1-balls on B_i, $\text{bd}(X) = \text{bd}(Y) = \{p, q\}$. If $\text{int}(X) \cap S = \emptyset$, then the segment $\sigma_T, T = \{p, q\}$ is α-exposed. Moreover, σ_T does not bound the interior of S_α (or, equivalently, σ_T is a* singular *simplex of the alpha-complex $\mathcal{K}_\alpha$).*

Proof: By Lemma 3.4 there exist two disks $D_{1,\rho}, D_{2,\rho}$ such that $p, q \in \text{bd}(D_\rho)$ and $D_{k,\rho} \cap B = X, k = 1, 2$.

Since $\text{int}(X)$ does not contain points of S, σ_T is α-exposed, and there are two weighted points x, y, $w_x = w_y = \rho^2$ that identify σ_T as α-exposed.

◇

Lemma 3.9 *There are at least three points of S on each B_i.*

Proof: B_i cannot have 0 points on it, because otherwise condition 2 would be violated for any $\text{int}(D_\rho)$ centered on B_i. Suppose B_i has only one point p of S. Then take D_ρ centered in p. By condition 1, D_ρ intersects B_i in a 1-ball I containing p. Then consider a point $c \in B_i - I$, and a disk D_ρ centered in c. Clearly this disk cannot contain p. Therefore, $\text{int}(D_\rho)$ does not contain ant point of S, violating condition 1. For the case of only two points of S on B_i one can repeat the reasoning in Lemma 3.4 and conclude again that condition 1 would not be satisfied. ◇

We are now ready to prove Theorem 3.1:

Proof: (i) $\mathcal{S}_\alpha(S)$ and B are homeomorphic.

By Lemma 3.9 there are at least three points of S on each connected component B_i of B. For each of these points, say p, there are exactly two other points of S on B_i, say q_1, q_2, such that the two 1-balls on B_i having p, q_k $(k = 1, 2)$ as boundary do not contain any other point of S. Therefore, by Lemmas 3.6-3.7, for each point of S there are exactly two incident 1-simplices in $\mathcal{S}_\alpha$. Observe that these segments cannot intersect each other in their interior. This could be easily proved here, but it will suffice to notice that the segments are part of the 1-skeleton of a simplicial complex. The α-exposed segments form a one 1-sphere for each component of B. We can then build a homeomorphism by mapping each segment $\sigma_T, T = \{p, q\}$ to the 1-ball $X \subset B_i$ that has p, q as boundary points and contains no other points of S.

(ii) $D(\mathcal{S}_\alpha(S), B) < \rho$.

Each segment $\sigma_T, T = \{p, q\}, p, q \in B_i$ of $\mathcal{S}_\alpha$ is mapped by the homeomorphism to a 1-ball $X \subset B_i$. This ball, by Lemma 3.5, is contained in the intersection of the two disks $D_{1,\rho}, D_{2,\rho}$, $p, q \in \text{bd}(D_{k,\rho}), k = 1, 2$ (see Figure 3). It is easy to see that for a point x in this intersection, the maximum distance δ to the closest point on the segment σ_T is $\delta < \rho$. Since this is true for all segments of $\mathcal{S}_\alpha$, the bound holds. ◇

Notice that locally the error bound can be made arbitrarily small. In fact, for each segment $\sigma_T, T = \{p, q\}$, if $||p - q|| = 2d$, the maximum local error is

$$\delta < \rho - \sqrt{\rho^2 - d^2}$$

which has limit zero as d tends to zero.

Therefore, while a ρ-dense sampling will suffice to reconstruct the manifold B with distance bounded by ρ, we can always make the approximation error arbitrarily small in any region $C \subseteq B$ by simply sampling C at a higher density. Also note that the expression for δ converges to zero quadratically, that is it is sufficient to double the density of the sampling to reduce the error by a factor of four.

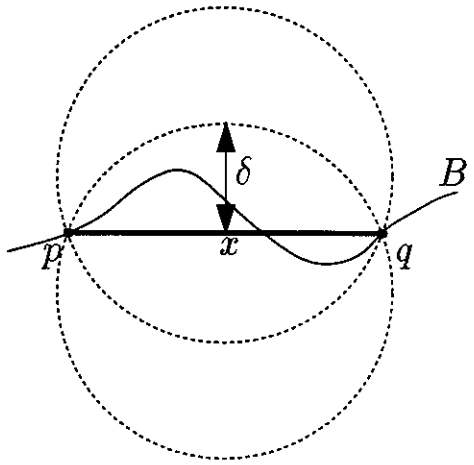


FIGURE 3: The maximum distance δ of a point x on the segment p, q to the closest point of B is bounded by ρ.

The conditions above restrict the domain of applicability of our reconstruction tool to surfaces whose radius of curvature is larger than ρ, as otherwise the ball-intersection requirement would be impossible to satisfy (see Figure 4). Note however the following: (i) This restriction parallels the band-limited requirement in Nyquist's sampling theorem; (ii) ρ can be made (at least in theory) arbitrarily small. The price to pay to reconstruct small-scale features is to use a high-density sampling, which is reasonable. On a more practical side: (iii) the sampling density of laser scanners is usually much smaller than object features of interest (otherwise large measurement errors would occur); (iv) points are not sampled on the sharp feature, but in its proximity; and (v) data collected in proximity of sharp (or high-curvature) features is usually subject to noise, and therefore not reliable. Accurately reconstructing sharp features (for example to segment the surface into a collection of smooth faces) requires an elaborate analysis of the data and/or additional knowledge of surface characteristics.

4 Conclusions

While the theorems above give us sufficient conditions for a sampling to allow a faithful reconstruction using α-shapes, in practice one has often to deal with less

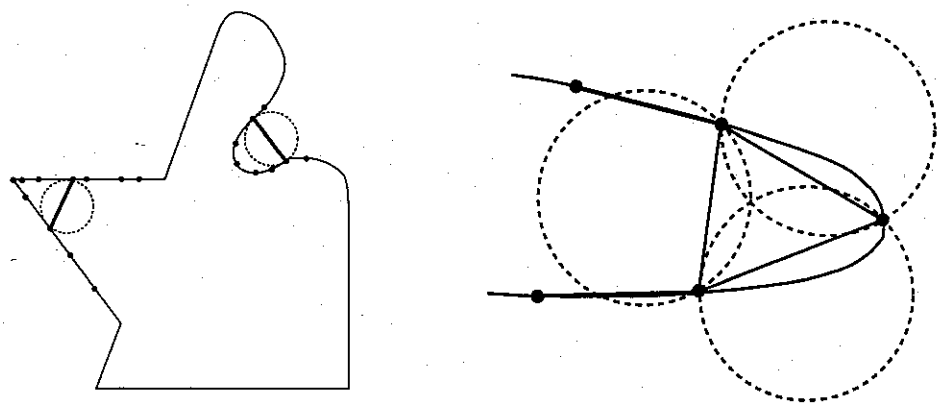


FIGURE 4: A small neighborhood of regions of curvature higher than ρ can be incorrectly reconstructed by the alpha-shape $\mathcal{S}_{\rho^2}$. Bold segments represent "extraneous" alpha-exposed 1-simplices. (a) A convex sharp feature and a concave high-curvature feature. (b) Extraneous alpha-exposed 1-simplex (detail).

than ideal scans.

In general, i.e. when the conditions of the theorems above are not satisfied, an alpha-shape is a non-connected, mixed-dimension polytope. We are interested in reconstructing *solids*, and therefore it is convenient to define a "regularized" version of an alpha-shape. The regularization should eliminate dangling and isolated faces, edges, and points from the alpha-shape, and recognize solid components.

In [2], we define a regularized alpha-solid,and describe an automatic method for the selection of an optimal α value, with a heuristic to improve the resulting approximate reconstruction in areas of insufficient sampling density. We are currently working on a proof for the 3D and general-dimension, weighted points version of the sampling theorem. Other directions for further research include efficient methods for the computation of two-manifold alpha-shape from the data points without computing the 3D Delaunay (or regular for the weighted case) triangulation. It would also be useful to develop a "real-time", incremental reconstruction methodology. With this approach, the partially reconstructed surface would be shown to the user as points get scanned.

Acknowledgments. Valerio Pascucci and Guglielmo Rabbiolo provided useful comments on the sampling theorem.

References

[1] BAJAJ, C., BERNARDINI, F., AND XU, G. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Computer Graphics Proceedings* (1995), Annual Conference Series. Proceedings of SIGGRAPH 95, pp. 109–118.

[2] BERNARDINI, F., BAJAJ, C., CHEN, J., AND SCHIKORE, D. Automatic reconstruction of 3D CAD models from digital scans. Tech. Rep. CSD-TR-97-012, Department of Computer Sciences, Purdue University, 1997.

[3] BOISSONNAT, J.-D. Geometric structures for three-dimensional shape representation. *ACM Trans. Graph. 3*, 4 (1984), 266–286.

[4] CHOI, B. K., SHIN, H. Y., YOON, Y. I., AND LEE, J. W. Triangulation of scattered data in 3D space. *Computer Aided Design 20*, 5 (June 1988), 239–248.

[5] CURLESS, B., AND LEVOY, M. A volumetric method for building complex models from range images. In *Computer Graphics Proceedings* (1996), Annual Conference Series. Proceedings of SIGGRAPH 96, pp. 303–312.

[6] EDELSBRUNNER, H. Weighted alpha shapes. Tech. Rep. UIUCDCS-R-92-1760, Department of Computer Science, University of Illinois, Urbana-Champagne, IL, 1992.

[7] EDELSBRUNNER, H., KIRKPATRICK, D. G., AND SEIDEL, R. On the shape of a set of points in the plane. *IEEE Trans. Inform. Theory IT-29* (1983), 551–559.

[8] EDELSBRUNNER, H., AND MÜCKE, E. P. Three-dimensional alpha shapes. *ACM Trans. Graph. 13*, 1 (Jan. 1994), 43–72.

[9] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. Surface reconstruction from unorganized points. *Computer Graphics 26*, 2 (July 1992), 71–78. Proceedings of SIGGRAPH 92.

[10] LORENSEN, W., AND CLINE, H. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics 21* (1987), 163–169.

[11] TURK, G., AND LEVOY, M. Zippered polygonal meshes from range images. In *Computer Graphics Proceedings* (1994), Annual Conference Series. Proceedings of SIGGRAPH 94, pp. 311–318.

[12] VELTKAMP, R. C. *Closed object boundaries from scattered points.* PhD thesis, Center for Mathematics and Computer Science, Amsterdam, 1992.

Periodic B-Spline Surface Skinning Of Anatomic Shapes

Fabrice JAILLET, Behzad SHARIAT and Denis VANDORPE
E-mail : {fjaillet,bshariat,vandorpe}@ligim.univ-lyon1.fr
Address: LIGIM, bat710, Université Lyon I
43 bd du 11 nov. 1918, 69622 VILLEURBANNE Cedex, FRANCE

1 Introduction

In the medical area, great improvements have been made in digital imagery techniques, as computed tomography (CT) or magnetic resonance imagery (MRI). Thus, more and more , the anatomic objects are described as a set of 2D cross-sections.

Here, we are interested in the reconstruction of the organs under investigation. This can be achieved by first extracting from each section a set of closed contours, corresponding to the intersection of the real surface with the sectional plane. Numerous methods have been presented, some of them based on "snakes" [CC90].

Next, the correspondence problem between linked contours on adjacent sections has to be solved [EPO91]. It means that the anatomic shapes have to be segmented into different branches. Then, the external shape has to be fitted with either an interpolation or an approximation surface.

A lot of methods have been proposed for the surface fitting problem. The most widespread approach is the polyhedral model. A set of triangular facets are generated between adjacent contours [FKU77, MSS92]. Since these methods produced a great number of triangles, some re-tiling algorithms have been developed [Tur92]. [JS95] proposes to use the triangulation as a guide to construct a tensor product smooth surface.

Skinning algorithms have been previously proposed for constructing a piecewise rectangular surface, but they present some limitations. In [PK96], the section curves are interpolated, this leads to a great number of control points to ensure the accuracy, while in [PT96] the contours are approximated but they only handle the case of open contours. Both methods are time consuming in order to bring the curves compatible for the skinning.

In this paper, we present a method for smooth closed surface approximation from 2D contours. Both the section curves and the skinned surface are approximated. This allows to reduce dramatically the amount of initial data while preserving the accuracy. Here, we are interested in single contour sections, but the handling of multiple contours is presented in the conclusion. Moreover, the resulting surfaces can easily be manipulated in an interactive system, with the help of the numerous techniques developed for the shape modification of B-spline curves and surfaces.

The organization of the paper is as follows. In section (2), some necessary B-spline formulas are briefly presented. In section (3), the general surface reconstruction method is presented, including a description of the curve fitting and the skinning of the section curves. The section (4) presents some experimental results. Finally the paper is closed by a conclusion section.

2 Closed B-Spline Formulas

A parametric k^{th} degree closed B-spline curve is a piecewise polynomial curve defined by:

$$C(t) = \sum_{i=0}^{m+k-1} P_{imod(m+1)} N^k_{imod(m+1)}(t) \qquad (1)$$

for any $t \in \mathbb{R}$, where the $m+1$ distinct P_i are the control points and the N_i^k B-spline basis functions of degree k can be recursively defined as:

$$N_i^0(t) = \begin{cases} 1, & \text{if } t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

$$N_i^k(t) = \frac{t - t_i}{t_{i+k} - t_i} N_i^{k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1}^{k-1}(t)$$

with 0/0=0, and defined on the knot vector t_i, which is an increasing sequence of knots.

For a closed B-spline curve, the knot sequence T has the following form:

$$T = \{\underbrace{t_{-k}, \ldots, t_{-1}}_{k}, \underbrace{t_0, \ldots, t_m}_{m+1}, \underbrace{t_{m+1}, \ldots, t_{m+k}}_{k}\}$$

$$\begin{aligned} t_0 &= 0 \text{ and } t_m = 1 \\ t_{-i} &= t_{-i+1} - (t_{m-i+1} - t_{m-i}) \\ t_{m+i} &= t_{m+i-1} + (t_i - t_{i-1}) \\ &\quad (\text{for } i = 1, \ldots, k) \end{aligned} \tag{2}$$

The previous definitions ensure that the curve is closed, $C(0) = C(1)$, and is defined for $t \in \mathbb{R}$ with a C^{k-1} continuity everywhere.

A biparametric closed B-spline surface can be built as a tensor product of closed B-spline curves. The surface defined here is closed in the contour's direction but open in the other one. A surface of degree $k \times l$ can be written as follows:

$$S(u,v) = \sum_{j=0}^{n} \sum_{i=0}^{m+k-1} Q_{imod(m+1),j} N^k_{imod(m+1)}(u) N^l_j(v)$$
$$\text{with } (u \in \mathbb{R}) \text{ and } (0 \leq v \leq 1) \tag{3}$$

where the $(m+1) \times (n+1)$ distinct control points $Q_{i,j}$ define the control net, and the N^k_i, N^l_j are the spline basis functions in the direction u and v respectively. The $N^k_i(u)$ are defined on a knot sequence $U = \{u_{-k}, \ldots, u_{-1}, u_0, \ldots, u_m, u_{m+1}, \ldots, u_{m+k}\}$ and the $N^l_j(v)$ are defined on the knot sequence $V = \{v_0 = \ldots = v_l, v_{l+1}, \ldots, v_n, v_{n+1} = \ldots = v_{n+l+1}\}$

The u parameter curve is a closed curve (in the direction of the contours), while the isoparametric curves in v direction are open (in the longitudinal direction).

3 B-Spline Surface Reconstruction

In this section, we describe the proposed method for the surface reconstruction from a set of planar cross-sections, each of them containing only one contour. First, we fit a closed B-spline curve to each polygonal contour. Then we perform the surface skinning of these section curves. This necessitates that all the section curves are defined on a common knot vector.

3.1 Closed curve fitting

The purpose of this section is the construction of a curve from the points of a planar contour. This problem has been widely studied in previous research works [Leo91, PT95]. We want to create a B-spline closed curve of degree k which approximates $p+1$ given contour points R_i. The number of control points, $m+1$, should be specified, as well as an appropriate knot vector $T = \{t_{-k}, \ldots, t_{-1}, t_0, \ldots, t_m, t_{m+1}, \ldots, t_{m+k}\}$. The degree and the number of control points could generally be refined using an iterative least-square approximation.

We can rewrite the equation (1) for each point R_i of the contour in matrix form: $[R] = [N][P]$, where $[R]$ is a vector of dimension $(p+1)$ representing the known contour points, $[N]$ is the B-spline basis coefficient matrix of dimension $(p+1) \times (m+1)$, and $[P]$ is the vector of dimension $(m+1)$ of the unknown control points of the B-spline curve.

Generally, the matrix $[N]$ is not square $(m < p)$ and the linear equations system can only be solved with a least-square approximation method [RF89]. The resolution of $[P] = [[N]^t[N]]^{-1}[N]^t[R]$ gives the control points which minimize the error on the curve.

For this, we should assign a parameter value $\bar{t}_i$ to each contour point R_i, which should take into account its position on the curve. The determination of this parameterization is very important in curve reconstruction [MK95], since a bad parameter value or knot vector can produce unwanted oscillations. The uniform parameterization is not recommended when the data are unevenly spaced, thus we prefer the chord-length method intensively employed in CAD.

The following equations [PT95] guarantee that every knot span $[t_i, t_{i+1}]$ contains at least one $\bar{t}_j$. This condition ensures that the system can be solved by Gaussian elimination without pivoting. The internal knots t_i for $i = 0, \ldots, m$ are defined as follows:
let $d = (p+1)/(m+1)$, then

$$\begin{cases} t_0 = 0 \,;\, t_m = 1 \\ t_i = (1-\alpha).\bar{t}_j + \alpha.\bar{t}_{j+1} \quad (i = 1, \ldots, m-1) \end{cases}$$

where $j = int(id)$ and $\alpha = id - j$.

3.2 Curve compatibility

To achieve the final surface reconstruction, we should fit a closed B-spline curve to each set of contour points. Then, due to its tensor product formulation, the surface skinning requires that all the $q+1$ curves have the same degree and defined on the same knot vector.

The first condition can be achieved easily by elevating the degree of each curve to the maximum of the degrees.

In return, it is not trivial to define a common knot sequence for all the curves, because the knot placement will affect the whole surface along the longitudinal direction. In [PT96], the authors have proposed an algorithm to merge the different knot vectors into a common one. But the resulting vector can contain thousands of points, necessitating a further treatment to eliminate a lot of knots while respecting a given tolerance. Since making the knot vectors compatible is very time consuming, if we assume that the contours are not too different in shape, we can simply average the internal knots:

$$u_j = \frac{1}{q+1}\sum_{i=0}^{q} u_j^i \qquad (j = 0, \ldots, m)$$

where the u_j^i are the j^{th} internal knot of the i^{th} curve. It is also assumed that all the $q+1$ intermediate curves have the same number of internal knots $(m+1)$. The complete knot sequence can be determined using equation (2). This produces generally satisfying values that reflect the distribution of the contour points. But some gaps in the knot sequence may appear when the points are unevenly spaced or the number of points is very different from a contour to another. We solve this problem by re-sampling adaptively new data points in these gaps.

Another requirement for the curve compatibility is the alignment of the given contours. Since a misalignment can produce twisted surfaces, the choice of a good alignment is fundamental for the quality of the resulting surface.

First of all, the contour points should be arranged in the same order, either clockwise or counterclockwise.

Next, the contours should be properly aligned. That means that a starting point should be found for each contour. The successive starting points form the longitudinal spine of the surface.

A good spine should be short in length and planar. We have developed an algorithm that computes the longitudinal spine according to the previous conditions, and taking into account the shape of the contours. The results of the choice of a misaligned and an aligned spine is shown on figure (1).

3.3 Surface skinning

We assume here that all the $q+1$ curves are defined on the same knot vector, have a common degree k and $m+1$ distinct control points ordered counterclockwise. We want to construct a skinned surface from this set of closed B-spline curves [Woo88, PK96, PT96]. The resulting biparametric surface should be

(a) Initial data (b) Twisted contours (c) Correct contours

Figure 1: Consequence of the choice of a longitudinal spine

closed in the contours' direction and open in the longitudinal one. Equation (3) can be rewritten for each curve C_h:

$$C_h(u) = S(u, v_h) = \sum_{i=0}^{m+k-1} \left\{ \sum_{j=0}^{n} Q_{imod(m+1),j} N_j^l(v_h) \right\} N_{imod(m+1)}^k(u)$$
$$\text{with } (u \in \mathbb{R}) \text{ and } (0 \leq v_h \leq 1)$$

To determine the points $Q_{i,j}$, representing the control net of the tensor-product surface, we fit an open B-spline curve to each column $(i = 0, \ldots, m)$ of the control points of the intermediate curves. By combining the previous equation with equation (1), we obtain:

$$P_{h,i} = \sum_{j=0}^{n} Q_{i,j} N_j^l(v_h) \qquad (h = 0, \ldots, q)$$

The determination of the control points of each longitudinal curve can be achieved by solving the matrix system $[P_i] = [N].[Q_i]$ and the final surface is obtained simply by joining the results together. The $[P_i]$ is a $(q+1)$ vector of the known data points representing the i^{th} control point on each section curve C_h, $[N]$ is the B-spline basis coefficient matrix of dimension $(q+1) \times (n+1)$, and $[Q_i]$ is the vector of dimension $(n+1)$ representing the unknown control points.

To solve these linear systems, we should assign a parameter to each data point $P_{h,i}$. To take into account the distribution of the points along the i^{th} longitudinal curve, we define a parameterization according to the chord-length method. To improve the final skinned surface, we determinate a common parameterization for all the columns by averaging the values:

$$\bar{t}_h = \frac{1}{m+1}\sum_{i=0}^{m} \bar{t}_h^i \qquad (h = 0, \ldots, q)$$

When $n = q$, the matrix $[N]$ is square and the $n+1$ control points can easily be obtained by $[Q_i] = [N]^{-1}.[P_i]$. This leads to the well-studied curve interpolation problem [RF89]. The N_i^k are defined on the common knot vector $\{u_i\}$ defined at section (3.2). The N_j^l should be defined on an appropriate knot vector $\{v_i\}$. To avoid any singularity in the system, we use the following technique of averaging the parameters:

$$v_0 = \ldots = v_l = 0 \text{ and } v_{n+1} = \ldots = v_{n+l+1} = 1$$

$$v_{j+l} = \frac{1}{l} \sum_{i=j}^{j+l-1} \bar{v}_i \quad (j = 1, \ldots, n-l)$$

With this method the knots reflect the distribution of the $\bar{t}_i$.

When $n < q$, the matrix $[N]$ is no more square, the system should be solved by a least-square approximation method. This leads to a great amount of data reduction and is worth to be used when the number of cross-sections is important. Similar to the knot placement of section (3.1), we define a knot vector for the v direction:
let $d = (q+1)/(n+1)$, then

$$v_0 = \ldots = v_l = 0 \text{ and } v_{n+1} = \ldots = v_{n+l+1} = 1$$

$$v_{i+l} = (1-\alpha).\bar{t}_j + \alpha.\bar{t}_{j+1} \quad (i = 1, \ldots, n-l)$$

where $j = int(id)$ and $\alpha = id - j$.

For both interpolating and approximating methods, the resulting smooth surface is C^{k-1} continuous in the u direction and C^{l-1} in the other direction.

3.4 Closing Surface

Since the initial data are structured as closed contours in one direction and open in the other direction, the isoparametric curves will consequently be closed in one direction and open in the other direction. However, this may be not satisfying for some anatomic shapes which are closed volumes, typically the bladder and prostate in our specific application.

One way to solve this problem is to cap the surface by a planar polygon but this leads to discontinuities in shape. Another way is to compute a smooth surface over the capping region and to ensure continuity with joining patches, as described in [Per92]. But this method requires sophisticated techniques, and moreover generates two additional patches.

Consequently, we have developed a simple solution to the closing problem. The idea is to extend the surface so that all the longitudinal curves converge to a same single point. This permits to model the whole object with a single B-spline surface. For this, we add a contour reduced to one point on top and bottom of the object, and we perform the previously described skinning method, on the new set of contours.

We first compute the gravity center of the two last (and first) polygonal contours G_{q-1} and G_q, and we take the symmetric of G_{q-1} about G_q, as shown on figure (2). Then, some constraints can be added at the chosen end-point to preserve the continuity without altering the shape of the reconstructed object. An example of closing is shown on figure (3).

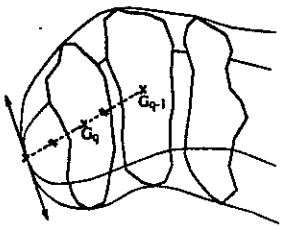


Figure 2: The closing method

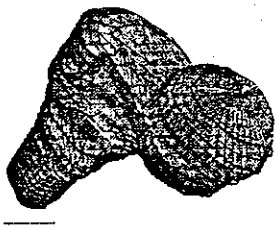
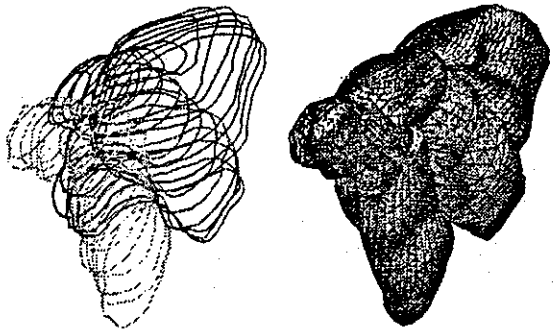


Figure 3: Closing of the femoral head

4 Results

The initial data are planar slices representing three internal neighbouring organs: rectum, prostate and bladder (Fig. 4(a)). We have reconstructed these objects with bicubic ($k = l = 3$) periodic B-spline surfaces with 15×10 control points. This has permitted to reduce dramatically the amount of the initial data, while producing C^2 continuous surfaces. A triangulated representation of the surfaces is shown in figure (4(b)).



(a) Initial data

(b) Periodic B-Spline surface models

Figure 4: Reconstruction of internal organs: rectum, prostate and bladder

We have also reconstructed lungs from particularly disturbed contours containing approximatively 5000 points (Fig. 5). The lungs have been modeled with periodic biquadratic B-spline surfaces with 30x15 control points. The results on figure (6) show the quality of the resulting surface.

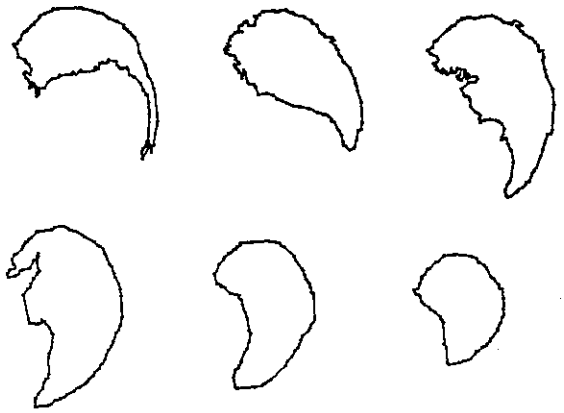


Figure 5: Some initial contours of the left lung

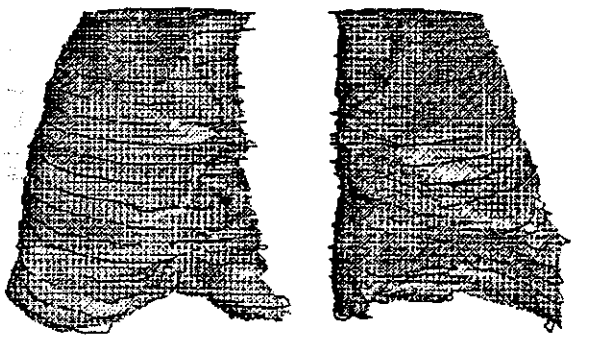


Figure 6: Initial contours and reconstructed surface of lungs

5 Conclusion

In this paper, an algorithm for the skinning of closed surfaces has been presented. The method generates quickly smooth surfaces with high continuity from planar closed contours. Interpolating can produce oscillating surface when successive contours have very different shapes. Thus, both the contour curves and the skinned surface are approximated, this leads to an important data reduction.

The methods proposed in previous research works [PK96, PT96] for curves compatibility, generate a great number of control points due to the accuracy and the smoothness requirements. This is normally followed by a very time consuming phase of reducing the number of knots of the common knot vector (about 90% of the total skinning time). Thus, we have proposed a simple averaging method to define the section curves on a common knot vector. The produced surface fit well the initial data, while keeping a small number of control points, and the physicians have been satisfied of the accuracy of the proposed model.

We have also proposed to close the surface by adding new contours to the top and bottom of the object. However, this method may create distorted surfaces as soon as the extremity contours are no more convex (or quasi convex). Thus we are currently working on extending the surface into an extremity curve whose shape is close to the planar threadlike skeleton of the last contour, rather than into a single point contour as proposed in section (3.4). But many continuity and curve compatibility problems arise.

Though, in this paper, we have studied the reconstruction of sections containing only one contour, it is feasible to modify the method in order to handle the case of multiple contours. Once we have solved the correspondence problem of the contours in adjacent sections, we reconstruct each branch separately. Then, the branches can be linked together with joining patches. We can use the techniques described in [Per92], in order to join the capping regions to the skinned surfaces. However, the case with many branches (four and more) is very complex and the continuity is difficult to ensure everywhere. The general case remains an open research area.

Finally, the proposed skinning method seems to be adequate and efficient enough to be integrated in an anatomical oriented interface (Fig. 7). As the precision of the reconstruction is not the main requirement, our application produces, at interactive time, realistic models of 3D closed shapes. These help visually the physicians during the radiotherapy treatment planning or iso-dose calculation.

Acknowledgments

We would like to thank the people of the NW Medical Physics of the Christie Hospital in Manchester,UK who provided us with the CT planar sections, and particularly Dr. C. Moore and Dr. R. MacKay for their contribution to this work.

This work is supported in part by the European BIOMED2 program.

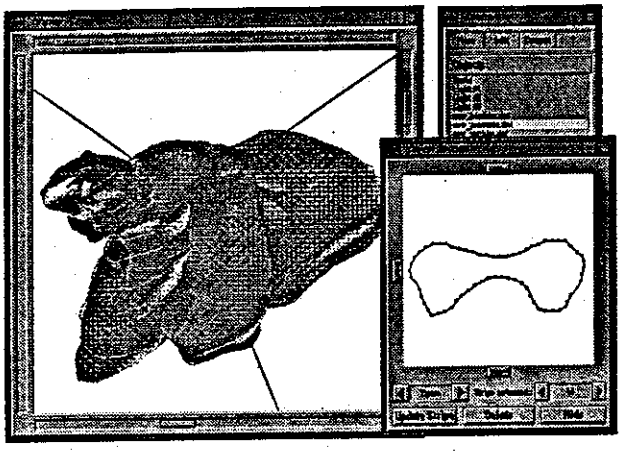


Figure 7: The interface

References

[CC90] L. D. Cohen and I. Cohen. A finite element method applied to new active contour models and 3D reconstruction from cross sections. In *Proceedings of Third International Conference on Computer Vision, Osaka, Japan*, 1990.

[EPO91] A. B. Ekoule, F. C. Peyrin, and C. L. Odet. A triangulation algorithm from arbitrary shaped multiple planar contours. *ACM Transactions on Graphics*, 10(2):182–199, April 1991.

[FKU77] H. Fuchs, Z.M. Kedem, and S.P. Uselton. Optimal surface reconstruction from planar contours. *Communications of the ACM*, 20(10):693–702, Oct. 1977.

[JS95] J. K. Johnstone and K. R. Sloan. Tensor product surfaces guided by minimal surface area triangulations. In *IEEE*, 1995.

[Leo91] J.-C. Leon. *Modélisation et construction de surfaces pour la CFAO.* Hermès, 1991.

[MK95] W. Ma and J. P. Kruth. Parameterization of randomly measured points for least squares fitting of B-spline curves and surfaces. *Computer-Aided Design*, 27(9):663–675, Sept. 1995.

[MSS92] D. Meyers, S. Skinner, and K. Sloan. Surfaces from contours. *ACM Transactions on Graphics*, 11(3):228–258, July 1992.

[Per92] E. Perna. *Modèles de surfaces pour la CFAO, raccordement de carreaux définis par produit tensoriel.* PhD thesis, Univ. Claude Bernard - Lyon I, Oct. 1992.

[PK96] H. Park and K. Kim. Smooth surface approximation to serial cross-sections. *Computer Aided Design*, 28(12):995–1005, Dec. 1996.

[PT95] L. Piegl and W. Tiller. *The NURBS book.* Springer, 1995.

[PT96] L. Piegl and W. Tiller. Algorithm for approximate NURBS skinning. *Computer-Aided Design*, 28(9):699–706, Sept. 1996.

[RF89] D. F. Rogers and N. G. Fog. Constrained B-spline curve and surface fitting. *Computer-Aided Design*, 21(10):641–648, Dec. 1989.

[Tur92] G. Turk. Re-tiling polygonal surfaces. In *Proceedings of SIGGRAPH '92*, pages 55–64. Computer Graphics, July 1992.

[Woo88] C. D. Woodward. Skinning techniques for interactives B-spline surface interpolation. *Computer-Aided Design*, 20(8):441–451, Oct. 1988.

Shape reconctruction using skeleton-based implicit surface

Serge PONTIER, Behzad SHARIAT and Denis VANDORPE
E-mail : {spontier,bshariat,vandorpe}@ligim.univ-lyon1.fr
Address: LIGIM, bât710, Université Lyon I
43 bd du 11 nov. 1918, 69622 VILLEURBANNE Cedex, FRANCE

April 24, 1997

1 Introduction

The problem of shape reconstruction of an unknown object, from a set of data points captured on its surface, has interested a lot of researchers. Some robust methods have been proposed for solid shapes using tensor product surface fitting or triangulation for example.
Moreover, some research works have been done on the reconstruction of deformable objects' shape as well as the simulation of their behaviour using physically based models. Particle systems and implicit functions are two examples of the most popular methods actually used. These methods are currently used in computer animation, medical applications and computer vision.

This paper presents a methodology for the reconstruction of human organ's form. The initial data is a set of stripes obtained by the digitization of CT scan-sections. To simplify the calculation, we have chosen to calculate separately the model of each section before to fit a three dimensional model through these sections.

Our aim is to produce a system which permits to model human's organs and to simulate their behaviour. Moreover, the calculated model should be computationally tractable and easily deformable. Therefore, we have chosen to employ implicit surfaces generated by skeletons. Thus, we will obtain a volume description of the organs, and we could use the method proposed by Gascuel [9] to deform them. In this formalism, an object O generated by a skeleton with a field function f is defined by :

$$O = \{P \in \mathbb{R}^3 / f(P) = iso\}$$

- iso is a given potential for an iso-surface points.
- f is a monotonically decreasing function of the distance from the point P to the skeleton.

Our work can be decomposed into three phases : the computation of a skeleton for each 2D section, the calculation of a 3D skeleton from these 2D skeletons and the definition of a scalar field function.

The second section of the paper discusses the state of the art of reconstruction techniques which use implicit functions. Section 3 explains our method and section 4 shows some results.

2 Related works

Several techniques have been published to present reconstruction techniques using implicit functions.

A first philosophy consists in extracting an iso-surface covering a set of data points, with the help of a marching cube algorithm. The difficulty is to be able to compute a potential at each point in the space. Wallin [20] uses the X-ray linear attenuation to assign a potential to any 3D point. Hoppe [10] uses a function, defined in a region close to the data, which estimates the signed geometric distance to the unknown surface. A similar approach, which has been introduced in [17] consists in reconstructing a cloud of points with an algebraic difference of two functions. One function defines an implicit object embedding the set of data points and the other is a volume spline, interpolating values of the first function in the data points.

In another approach, the shape of an unknown object is calculated by the deformation of a primitive to fit the data points. Pentland [16] and Bajcsy [1] are the firsts who have employed the superquadrics [3] for the reconstruction. Later, the deformable superquadrics have been introduced by Terzopoulos [18]. The reconstruction method proposed by Terzopoulos consists in applying some external forces to deform the superquadrics. The Bajcsy's and Terzopoulos' methods have been improved by several

works [11, 15, 2]. Similarly, a methodology of 2D contour reconstruction using snakes [7] and another using hyperquadrics [6] have been proposed.

A last class of reconstruction methodologies uses implicit functions generated by skeletons. Most of them work with punctual skeletons, which generate spheres, and use an energy minimization process. Miller [13] uses a ball which grows as far as it fits the cloud of points. Muraki [14] employs a union of blobs to model a set of data points. The blobs are subdivided and moved to minimize an energy until they reach the desired precision. This process has been sped up in [19] then [4]. In order to avoid an iterative process, Lim [12] describes an object with a union of spheres which are obtained from the Delaunay graph of the data points. Lastly, Ferley [8] reconstructs a cloud of points with a skeleton composed of several B-spline curves which are surrounded with a non-uniform field. Each curve generates an implicit surface and the object is modeled with a blended implicit surface.

All these methodologies are difficult to be employed in our system. The iso-surface can not easily be deformed. Moreover, the methods using the superquadrics need a formal definition of deformations and they are slow. Finally, the union of primary elements generates a piecewise function for which the deformation is difficult to realize and is time consuming.

3 A reconstruction method for 2D sections

Here, we propose a methodology permitting the calculation of a human's organ model from a set of planar data sections. To simplify the obtained model and to reduce the calculation time of the simulation, we have chosen to use an implicit function based on skeletons. In order to increase the efficiency of our algorithm, we do not model an object with a blended iso-surface. Indeed, in our method, although the skeleton is composed of a set of skeletal elements, the scalar field function remains identical for the whole skeleton; therefore no blending is performed and the objects can easily be modeled with a distance surface.

Rather than calculate directly a 3D skeleton, we prefer to take into account the structure of the input data. Thus, we create 2D skeletons, one per stripe, and later a 3D skeleton is computed from the 2D ones.

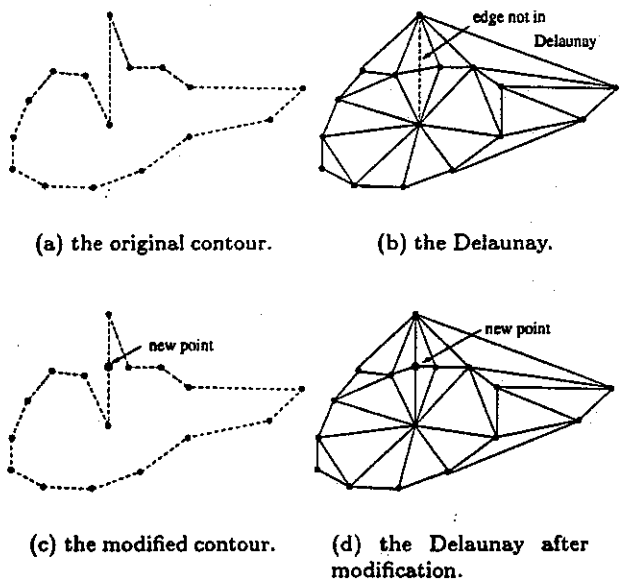
3.1 Computation of a 2D skeleton

A skeleton which is a coarse description of a shape is computationally intractable. In fact, its calculation is based on an erosion process which is time consuming. In order to avoid this, a weighted threadlike skeleton surrounded by an uniform field is used.

The threadlike skeleton is first calculated with the help of Voronoï graph. To each vertex of this graph, we associate a weight which is calculated according to the distance between digitized points and the skeleton.

3.1.1 The Delaunay graph computation

The first step consists in computing the Delaunay graph from the set of data points. At this level, several problems should be solved. First, the graph must contain all the contour's edges. When it is not the case some points are created in the middle of the edges which are missing in the Delaunay graph (fig. 1).



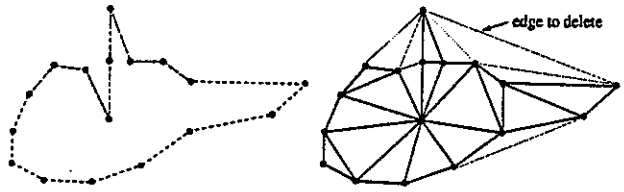
(a) the original contour. (b) the Delaunay.

(c) the modified contour. (d) the Delaunay after modification.

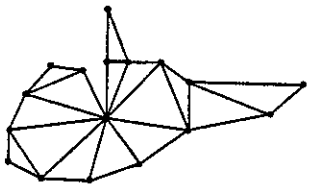
Figure 1: The contour's modification when an edge does not appear in the Delaunay graph.

The Delaunay graph generates a convex hull of the data points (fig. 2). So some triangles must be removed to keep only those which are inside the object. This process consists in deleting recursively all the triangles which have a boundary edge which does not belong to the initial contour.

Finally, to avoid external skeleton, some new points are created on the edges which are the hypotenuse

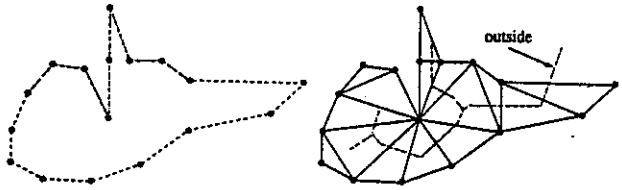


(a) the original contour. (b) the original Delaunay.

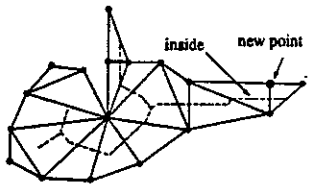


(c) the Delaunay without external triangles.

Figure 2: Suppression of some Delaunay triangles in concave area.



(a) the original contour. (b) the Delaunay and the threadlike skeleton.



(c) the Delaunay and the threadlike skeleton after the contour's modification.

Figure 3: Skeleton external element suppression.

of obtuse Delaunay triangles (fig. 3). These new points are the projection of the third vertex of the corresponding triangle.

This step is based on the Boissonnat's work [5].

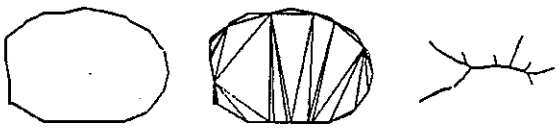
3.1.2 The computation of the threadlike skeleton

The threadlike skeleton of a set of points is given by the internal elements of the Voronoï graph. It is computed from the Delaunay graph. It is defined by a set of vertices (gravity center of Delaunay triangles) and a set of edges (connecting the vertices). The main disadvantage of this method is its sensitivity to noise. Indeed, frequently, the obtained skeleton contains a lot of branches (fig. 4).

The skeleton's branches are created by some small triangles on the boundary. Our simplification scheme consists in removing the triangles which have two boundary edges and whose area is smaller than a threshold (fig. 5). This implies that the generated iso-curve does not fit to every digitized points : some of them are eliminated during this process.

3.1.3 The weighted skeleton

The obtained threadlike skeleton is centered within the cloud of points. Consequently, the distance to the skeleton is not the same for each 3D digitized points. If we use this skeleton a non-uniform field function should be employed. To prevent this, we propose to

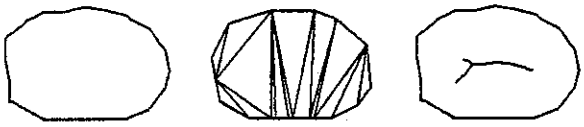


(a) the data points (b) the non-simplified Delaunay graph. (c) the corresponding skeleton.

Figure 4: A noisy skeleton.

build a "weighted skeleton" which is a non-uniform offset of the previous threadlike skeleton. We model this offset by weights. In order to obtain the coarse description with the smallest size, at least one of the skeleton's vertices should have a null weight. In other words, the object is eroded so far as to touch the threadlike skeleton at this vertex.

We calculate only the weights of Voronoï vertices belonging to the threadlike skeleton. The weight of other skeleton's points is approximated. The weights' computation is achieved using the Delaunay graph :



(a) the data points

(b) the simplified Delaunay graph.

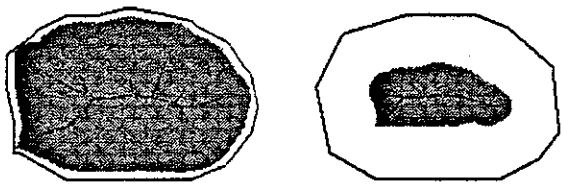
(c) the corresponding skeleton.

Figure 5: A simplified skeleton.

$$\omega(P) = \begin{cases} dist(P, Proj_E) - d_{min} \\ \quad \text{if } P \text{ is the gravity center} \\ \quad \text{of a triangle which has only} \\ \quad \text{one boundary edge } E \\ \\ dist(P, S) - d_{min} \quad \text{otherwise} \end{cases}$$

where :

- $\omega(P)$ is the weight of the Voronoï vertex P of the skeleton.
- E is the only boundary edge of a triangle.
- $Proj_E$ is the projection of P on E.
- S is a Delaunay vertex.
- d_{min} the smallest distance from a digitized point to the threadlike skeleton.



(a) without skeleton simplification

(b) with skeleton simplification

Figure 6: The coarse description and the deformation zone.

3.2 The field function

The potential of a 3D point P is given by :

$$F(P) = f(dist(P, Q) - \omega(Q)),$$

where :

- Q : is the projection of P on the skeleton.
- f : is a monotonous decreasing function.

In order to obtain a weight for all the skeleton points, a linear approximation has been made between two successive vertices. As a linear approximation is used, the generated surface is not really smooth for the points projected on a Voronoï vertex. To solve this problem, the weights could be interpolated with B-spline functions.

4 Results

In figure 7, a complex contour of a lung is reconstructed with the help of the proposed algorithm. This shows the efficiency of our method.

(a) a contour of a lung

(b) the non-simplified skeleton

(c) the weighted non-simplified skeleton

(d) reconstruction with the non-simplified skeleton

(e) the simplified skeleton

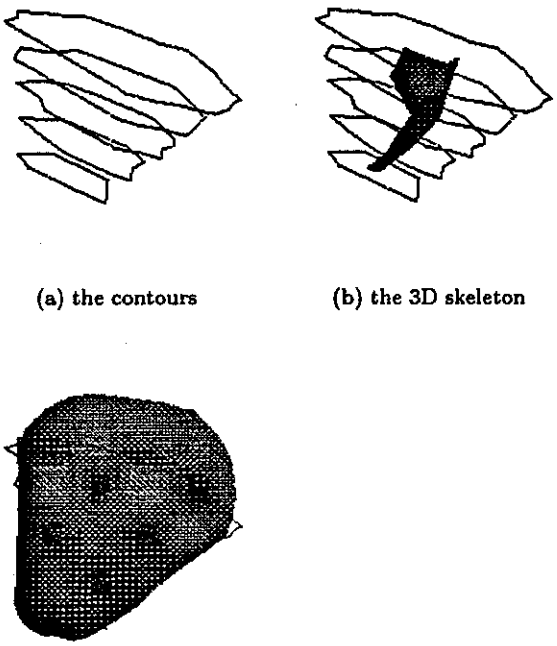
(f) the weighted simplified skeleton

(g) reconstruction with the simplified skeleton

Figure 7: An example of 2D reconstruction.

3D shapes can also be reconstructed as shown in figure 8. For this, adjacent skeletons are connected by triangles. Thus, a weighted 3D surface skeleton is obtained. We have solved this 3D skeleton problem

for simple cases, but some specific problem have to be solved for the general case.



(a) the contours (b) the 3D skeleton

(c) the 3D reconstruction

Figure 8: An example of 3D reconstruction.

5 Conclusion and future work

In this paper, we have presented a methodology permitting to reconstruct the shape of 2D contours, using the skeleton based implicit function formalism. For this, we have proposed an algorithm to compute a weighted threadlike skeleton which gives a coarse description of the shape. Although the skeleton is constituted of a set of skeletal elements, an identical field function is used for the whole skeleton resulting in a non-blended shape. Finally, some examples of the algorithm's extension to 3D space have been given.
This algorithm permits to have a model with variable complexity. If a precise reconstruction is needed, no skeleton's simplification will be performed. The shape will be described with a complex threadlike skeleton providing a very large coarse description and a narrow deformable zone. Consequently, the deformations will be limited. If a high precision reconstruction is not the main requirement, a very simplified threadlike skeleton can be used and the deformable zone is large. So important deformations could be performed.

Our future work includes the improvement of the reconstruction of 3D shape. Indeed, our methodology works with simple objects, but some problems can arise for objects with a complex skeleton.
In this paper, we have shown that sections containing several contours can be treated easily. Nevertheless, sections with holes are not automatically reconstructed : at least one Delaunay triangle must be manually removed to obtain the correct reconstruction. We are currently studying an approach permitting to suppress this step.
Finally, some studies can be made on the deformation process. Indeed, if the model undergoes some inelastic deformations, it could be attractive to modify the skeleton's weights to change the shape of the object.

References

[1] R. Bajscy and R. Solin. Three-dimensional object representation revisited. *IEEE First Conference on Computer Vision*, pages 231–240, 1987.

[2] Eric Bardinet, Laurent D. Cohen, and Nicolas Ayache. A parametric deformable model to fit unstructured 3d data. Technical Report 2617, INRIA, July 1995.

[3] A. H. Barr. Superquadrics and angle preserving transformations. *IEEE Computer Graphics Apllication*, 1:11–23, 1981.

[4] Eric Bittar, Nicolas Tsingos, and Marie-Paule Gascuel. Automatic reconstruction of unstructured 3d data : Combining a medial axis and implicit surfaces. *Eurographics'95*, Sept 1995.

[5] J. D. Boissonnat and B. Geiger. Three dimensional reconstruction of complex shapes based on the delaunay. Technical Report 1697, INRIA, April 1992.

[6] Isaac Cohen and Laurent Cohen. A hybrid hyperquadric model for 2-d and 3-d data fitting. Technical Report 2188, INRIA, January 1994.

[7] B. Derdouri, M. Neveu, and D. Faudot. Reconstruction d'objets 3d déformables. *Actes de GROPLAN 92, Nantes*, pages 99–106, Novembre 1992.

[8] Eric Ferley, Marie-Paule Gascuel, and Dominique Attali. Skeletal reconstruction of branching shapes. In *Implicit Surfaces'96 : 2nd International Workshop on Implicit Surface*, Eindhoven, The Netherlands, October 1996.

[9] Marie-Paule Gascuel. An implicit formulation for precise contact modeling between flexible solids. *Computer Graphics Proceedings*, 27:313–320, August 1993.

[10] H. Hoppe, T. DeRose, T. Duchamp, J. McDonnald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computeur Graphics*, 26(2):71–77, July 1992. notes of SIGGRAPH'92.

[11] Aleš Leonardis, Franc Solina, and Alenka Macerl. A direct recovery of superquadric models in range images using recover-and-select paradigm. In *Proceeding of third European Conference on Computer Vision*, Stockholm, Sweden, May 1994.

[12] Chek T. Lim, George M. Turkiyyah, Mark A. Ganter, and Duane W. Stroti. Implicit reconstruction of solids from cloud point sets. In *Solid Modeling '95*, pages 393–402, may 1995.

[13] James V. Miller, David E. Breen, William E. Lorensen, Robert M. O'Bara, and Michael J. Wozny. Geometrically deformed models : A method for extracting closed geometric models from volume data. *Computer graphics*, 25(4):217–226, July 1991.

[14] Shigeru Muraki. Volumetric shape description of range data using "blobby model". *Computer Graphics*, 25(4):227–235, July 1991.

[15] Jinah Park, Dimitri Metaxas, and Alistair Young. Deformable models with parameter functions : Application to heart-wall modeling. In *Conf. on Computer Vision and Pattern Recognition*, pages 437–442, June 1994.

[16] Alex P. Pentland. Perceptual organization and the representation of natural form. *Artificial Intelligence*, 28(3):293–331, 1986.

[17] Vladimir V. Savchenko, Alexander A. Pasko, Oleg G. Okunev, and Tosiyau L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, 1995.

[18] Demetri Terzopoulos and Dimitri Metaxas. Dynamic 3d models with local and global deformations : Deformable superquadrics. *IEEE Transactions on pattern analysis and machine intelligence*, 13(7):703–714, July 1991.

[19] Nicolas Tsingos, Eric Bittar, and Marie-Paule Gascuel. *Implicit Surfaces for Semi-automatic Medical Organ Reconstruction*, chapter Techniques in Modelling Virtual Environments, pages 3–15. Academic Press, Leeds, UK, juin 1995.

[20] A. Wallin. Constructing isosurfaces from ct data. *IEEE CG and A*, 11(6):28–33, November 1991.

Dynamizing Domination Queries in 2-dimensions: The Paper Stabbing Problem Revisited

Michael G. Lamoureux, J.D. Horton, and Bradford G. Nickerson

Abstract:

This paper describes a $\Theta(n)$ space data structure, the domination map, which supports domination queries on 2-dimensional data points in $\Theta(\lg n + t)$ time, for t points in range. If a modified k-d tree structure, described within, is used to index the domination map, then it may be dynamically updated, with insertions and deletions requiring $\Theta(\lg^2 n)$ amortized time, when $O(n \lg n)$ storage is permitted to store the modified 2-d tree index structure.

An extension to the domination map that supports domination queries on 3-dimensional data points, without increasing the amount of storage required or the amount of query time needed, is described. It supports dynamic updates as well when indexed with a modified 3-d structure, and these also require $\Theta(\lg^2 n)$ amortized time.

Introduction

Let P be a set of points in a 2-dimensional Euclidean space E^2 and let D be the domain of all subsets of P defined by distinct semi-infinite range queries specified by the Cartesian product of two semi-infinite ranges of the form $[q^i,\infty]$. In this paper, superscripts refer to coordinates of a point and subscripts refer to distinct points. Domination searching with respect to P and D refers to the task of processing P so that for any semi-infinite range query q in D, the subset of points in P that lie in q can be determined.

The performance of a data structure for domination search is measured based on S(n), the required storage, and Q(n), the required query time. Let P_q ($P \cap q$) denote the set to be computed. Two classes of domination search are distinguishable. In count mode, it is only necessary to compute the cardinality of P_q, and in report mode, it is necessary to determine every element of P_q. This paper assumes that the queries must be answered in report mode.

The existence of efficient domination search algorithms ([Chaz86],[Chaz87]) motivates the following questions: (1) how efficiently can a domination query be solved in the worst case in a given space complexity?, (2) how efficiently can a domination query be solved if only linear storage is available, and (3) how efficiently can a domination query be solved when the solution must be dynamic?

This paper develops a $\Theta(n)$ space data structure that solves domination queries on 2 and 3 dimensional data points in $\Theta(\lg n + t)$ time, for t points in range, while allowing for dynamic updates in $\Theta(\lg^2 n)$ amortized time when $O(n \lg n)$ storage is permitted for an index structure. This compares well to the work of [Chaz87] who presents static linear space algorithms for domination search on 2-d data points in $\Theta(\lg n + t)$ time and on 3-d data points in $\Theta(\lg^2 n + t)$ time. By allowing an additional factor of $O(\lg n)$ for storage, there is a dynamic solution which is of the same complexity for searches in 2-d and faster for searches in 3-d.

The results contained in this paper are based on an extension to an optimal solution of the 2-d paper stabbing problem discussed in [Chaz87]. The paper stabbing problem is defined as follows: suppose that you have n sheets of paper attached to one corner of your desk; none of them completely hidden behind any other. A query comes as a needle through the first t sheets at an arbitrary point on the desk.

The solution of [Chaz87] is based on a priority search directed acyclic graph (DAG) structure which is described below. A description of the domination map, which extends the priority search DAG with a polytope index, and a k-d tree ([Bent75]) face index for dynamic updates, follows.

In the initial description of the domination map, it is assumed that the data set is static and that preprocessing only needs to be done once. This supposes that the cost of the preprocessing operation can be amortized over many queries and is thus negligible. Afterwards, the domination map is dynamized.

The Priority Search DAG

Consider the point $p = (p^x,p^y)$ in the Euclidean plane E^2. A point p_1 dominates a point p_2, denoted $p_2 \angle p_1$, if $p_2^x \angle p_1^x$ and $p_2^y \angle p_1^y$. Let $P = (p_1,p_2,...,p_n)$ be a sequence of points of the form $p_i = (p_i^x,p_i^y)$ satisfying the appearance property: for any i,j the relation $p_i \angle p_j$ implies $i < j$.

The appearance property is equivalent to topological sorting and it informally stipulates that each piece of paper is at least partially visible for any point q in E^2. Define P_q as the set of points in P dominating q. The paper stabbing problem becomes: process P so that for any q in E^2, the set P_q can be computed.

The priority search DAG is constructed as a planar graph consisting of the visible parts of the rectangles. Without a loss of generality, the origin is chosen such that all points are in a bounded northeast quadrant and, for convenience, all p_i^x (and p_i^y) are distinct. The priority search DAG is defined as the isothetic planar subdivision (all boundaries are parallel to an axis) obtained as follows: for each i = 1,...,n in turn, extend a horizontal segment p_ih_i leftward and a vertical segment p_iv_i downward from p_i until a segment or an axis is reached. The point p_i is the primary anchor point of its incident edges and the points h_i and v_i are supporting points. When the p_i^x and p_i^y are not distinct, one vertex functions as two or more of $\{p_i,h_i,v_i\}$. An example of a priority search DAG is given in Figure 1.

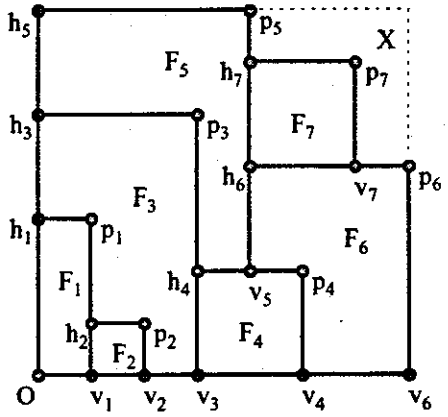


Figure 1: An example of a priority search DAG.

A domination query for an arbitrary point q is answered as follows: the DAG is processed for the efficient retrieval of a face containing the point q in logarithmic time. [Chaz87] indicates numerous methods. The points in range are located by locating all the edges intersecting the axis of the search region defined by q. (This may be accomplished with an appropriate graph index or line intersection algorithm.) These are the starting edges of the query which yield the starting vertices (the first point located along the edge in the positive direction). All positive edges from the starting vertices, and from all other points found in range, are traversed until all of the vertices in range have been encountered, yielding all points in range. [Chaz87] has shown that this can be done in O(lg n + t) time, which is optimal.

The polytope index used in 2-d is the following, developed to work efficiently with the modified k-d tree structure which is used to dynamize the structure. It is a map index which is somewhat reminiscent of a DCEL which indexes and keeps track of required indexing information on all of the map elements: vertices, edges, and faces. It consists of three tables assumed to be accessible in constant time given an appropriate unique element identifier.

The first table keeps track of all the vertices and records the unique identifier, coordinates, type, and incident edges of each vertex. The second table keeps track of all the edges and records the unique identifier, type, endpoints, and adjacent faces of each edge. The third table keeps track of all the faces and records the unique identifier and defining edges of each face.

Vertices are of two types: primary, the actual data points, and secondary, the other points in the priority search DAG. If the edge is horizontal (x), the face below and the face above form the adjacent faces, if the edge is vertical (y), then it is the face to the left and the face to the right. Only four edges are stored to define a face, even if more exist. The two edges that connect to the primary anchor point in the upper right corner, defining the top-most and right-most edges, and the two edges that connect to the left end point of the leftmost edge on the bottom of the face, defining the left-most and the bottom-most edges. An edge is primary if it is incident with a primary vertex, it is secondary otherwise.

For the priority search DAG given in Figure 1, the map index is given in Appendix 1.

The Domination Map

The domination map is the data structure formed from the union of the priority search DAG of [Chaz87], the map index described above, and the modified k-d tree structure used to index the faces of the priority search DAG structure. The modified k-d tree structure is constructed as a height balanced structure of $\Theta(\lg n)$ levels in the static case, and has $O(\lg n + \lg\lg n)$ levels in the dynamic case.

The node structure of the modified k-d tree contains the following information:
a) the data point (real or virtual) used to split the space
b) the dimension being discriminated on (chosen to minimize tree height in the dynamic case)

The idea is the following: assuming a bounded space, each branch defines a well defined division of the current quadrant into two smaller quadrants. These can be easily computed on the way down the tree. The tree is constructed on a static data set by using the primary points and secondary points as necessary to repeatedly divide the space into quadrants until each quadrant is contained in only one face of the priority search DAG. By choosing the primary and secondary points to divide the space, there will be at most 4n quadrants after the division. Using a modification of the algorithm to construct a balanced k-d tree guarantees a maximum tree height of $\Theta(\lg n)$.

Figure 2 illustrates a height balanced k-d tree which may be used to index the priority search DAG of Figure 1 and Figure 3 illustrates the division of space. In Figure 1, the discriminators alternate between x and y down the tree and the labels on the edges represent the division of space effected by the node.

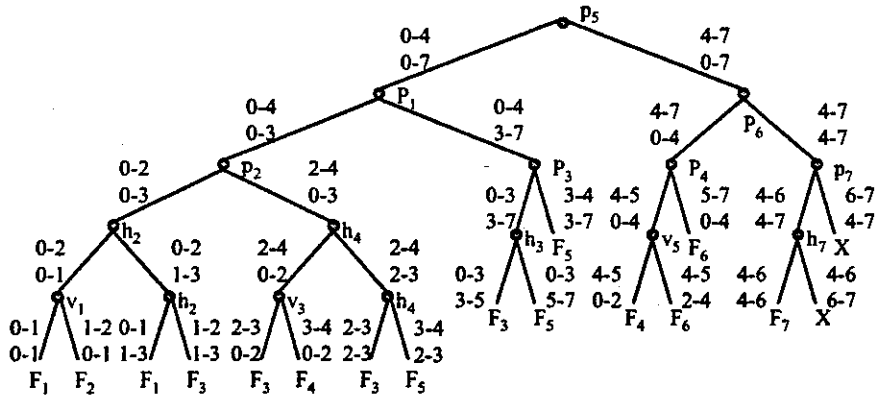


Figure 2. The height balanced k-d tree which is used to index the priority search DAG of Figure 1.

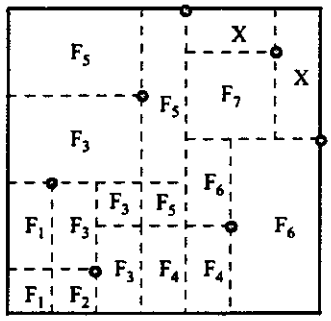


Figure 3. Division of space effected by the k-d tree of Figure 2.

In order to effect a domination search on the domination map, it must maintain the underlying priority search DAG structure - each rectangle must be partially visible. Note that for a point p_j not in the structure, and distinct from all p_i in the structure, extending lines p_jh_j and p_jv_j in the priority search DAG from the point p_j until edges or axes are encountered will produce a new priority search DAG such that all rectangles are partially visible. Thus, an insertion can be accomplished if the k-d tree and map indexes can be updated efficiently.

Dynamizing the Domination Map

The map index can be updated by using the k-d tree to find the face F_j in which the point p_j exists. The dropped segments intersect (1) the left-most and bottom-most edges of the face or (2) left and bottom edges of the face defined by top-most and right-most edges of faces defined by anchor points that are dominated by the anchor point of the face F_j. In the second case, these edges can be found in logarithmic time by checking at most O(lg n) quadrants to the left and at most O(lg n) quadrants down.

Once the edges have been found, the following updates are made to the map index where the left edge is denoted L=(LL,LR) with left and right faces LLF, LRF and type Lt, and the bottom edge is denoted B=(BL,BR) with bottom and top faces BLF, BRF and type Bt, and where the new face is denoted NF. Note that each update can be done in constant time.

Update the edge table as follows:

Add: $[(h_j,p_j),p,h_j,p_j,NF,LRF]$ Add: $[(vj,pj),p,vj,pj,NF,BRF]$

Add: $[(LL,h_j),s,LL,h_j,LLF,NF]$ Add: $[(h_j,LR),Lt,h_j,LR,LLF,LRF]$ Del: (LL,LR)

Add: $[(BL,v_j),s,BL,v_j,BLF,NF]$ Add: $[(v_j,BR),Bt,v_j,BR,BLF,BRF]$ Del: (RL,RR)

Update the vertex table as follows:

Add: $[p_j,(p_j^x,p_j^y),p,(h_j,p_j),-,(v_j,p_j),-]$ Add: $[h_j,-,s,-,(h_j,p_j),(LL,h_j),(h_j,LR)]$ Add: $[v_j,-,s,(BL,v_j),(v_j,BR),-,(v_j,p_j)]$

Update: LL [y-RE $\rightarrow (LL,h_j)$] Update: LR [y-LE $\rightarrow (h_j,LR)$]

Update: BL [x-RE $\rightarrow (BL,v_j)$] Update: BR [x-LE $\rightarrow (v_j,BR)$]

Update the face table as follows:

Add: $[F_j,(LL,h_j),(v_j,p_j),(LL_{x\text{-}RE}),(h_j,p_j)]$

The next step is to update the k-d tree index structure. This is done by dividing each quadrant intersected by a dropped segment into two, or three if it is the quadrant that contains the anchor point, quadrants. Since at most O(lg n) quadrants are intersected, and each may be split in constant time, the time to accomplish a single insertion is O(lg n) and the worst case storage will be O(n lg n) as O(lg n) points may need to be inserted into the k-d tree index to accomplish an insertion.

In a sequence of insertions, it is possible that the k-d tree structure could become unbalanced with the height increased beyond O(lg n). This is avoided by rebuilding a subtree whenever one side becomes twice as deep as the other side. With calculations, it can be shown that the entire tree will not have to be rebuilt more than once in a sequence of O(n) insertions, and subtrees at level x will not have to be rebuilt more than $O(2^x)$ times in a sequence of O(n) insertions. This allows the total rebuilding time over a sequence of n insertions to be bounded at $O(n\, lg^2 n)$, bounding the amortized insertion time at $O(lg^2 n)$.

Deletions are effected as follows. The vertex p_j, and its edges x-LE and y-LE, are removed. If the left vertex of x-LE does not have an incident edge downward, it is removed and its upward edge is extended downward until a segment (axis) is intersected, and its leftward edge (and leftward incident vertex) is also removed, otherwise it is simply removed. If the bottom vertex of y-LE does not have an incident edge leftward, then it is removed and its rightward edge is extended until a segment (axis) is intersected, and its downward edge (and downward incident vertex) is also removed, otherwise it is simply removed.

If the removal of an edge causes an attached supporting point to have two edges, it is removed as well and one incident edge (right or top) is extended until a supporting segment is reached. In the worst case, O(n) supporting vertices will be removed, and thus the worst case deletion will require O(n lg n) time to complete as the k-d tree index will have to be completely rebuilt. However, this may only occur once in a sequence of O(n) deletions. It is also the case that the xth worst deletion can only occur $O(2^x)$ times in a sequence of O(n) deletions, and this will require rebuilding of the k-d tree on the order of $O(n/2^x)$ time, giving a worst case rebuilding time of $O(n lg^2 n)$ over a sequence of O(n) deletions. This bounds the amortized deletion time at $O(lg^2 n)$.

Extending the Domination map to 3 Dimensions

The priority search DAG can be extended as follows: assume all points lie in a bounded, northeast quadrant and drop segments towards each bounding plane until either a face of another 3-d semi-orthogonal region is hit or the bounding plane is reached. From each of these points, drop 2 edges towards the other bounding planes until edges of a face of a 3-d region are hit or the axis is reached. The priority search DAG now has 6 supporting points and nine primary and secondary edges for every point in range. Thus, it requires $\Theta(n)$ storage.

The polytope index is extended as follows and requires $\Theta(n)$ storage:
1) The vertex index is extended to keep track of the left and right incident edges parallel to the z axis
2) The edge index is extended to keep track of the left and right incident faces in both relevant planes
3) The face index keeps track of the left and right 3-d semi-orthogonal regions.
4) A region index is added and a region is defined by a unique region identifier, a left, right, top, bottom, front, and back face.

The k-d tree is built analogously to the 2-d version, with the only difference that it may require up to 8n octants to ensure that an octant only indexes one region. On a static data set, it requires $\Theta(n)$ storage.

Thus, in the static case, the structure requires $\Theta(n)$ storage. Domination queries require $\Theta(\lg n + t)$ as they can be performed as follows. Use the k-d tree to locate the region that contains the point q in $\Theta(\lg n)$ time. Use the polytope index to progress rightward, topward, and forward in the direction of the axis to locate all regions that intersect the axis of the query region. This provides the faces that intersect the query region, and thus the edges. With the starting edges, locate the starting points from which all rightward, topward, and forward incident edges are traversed until all vertices in range are located.

It can be dynamized analogously to the 2-d case, and although the algorithms are more involved, the update and storage times remain the same with storage bounded by $\Omega(n)$ and $O(n \lg n)$.

Conclusions

The domination map is a very efficient data structure for domination queries on 2 and 3 dimensional data points, especially in the dynamic case. In the static case, it is as efficient as the structure of [Chaz87], on which it builds, for 2-d data and more efficient than the structure of [Chaz87] for 3-d data. In the dynamic case, it retains the same query times for the small cost of an additional $O(\lg n)$ factor of storage in the worst case while permitting amortized dynamic updates of $O(\lg^2 n)$.

Further work will involve an attempt to extend the structure to handle k-dimensional data and improve the update algorithms on the modified k-d tree so that only a constant number of regions need to be split without increasing the overall amortized rebuilding time, thus eliminating the additional $O(\lg n)$ storage.

References

[Bent75] Bentley, Jon Louis. "Multidimensional Binary Search Trees Used for Associative Searching", Communications of the ACM, Volume 18, Number 9, September 1975, pp. 509 - 517.

[Chaz86] Chazelle, Bernard. "Filtering Search: A New Approach to Query-Answering", SIAM J. Comput., Vol. 15, No. 3, August 1986, pp. 703 - 724

[Chaz87] Chazelle, Bernard, and Edelsbrunner, Herbert. "Linear Space Data Structures for Two Types of Range Search", Discrete Computational Geometry, Vol. 2, pp. 113 - 126, 1987

Appendix 1

The polytope index for Figure 1.

Table 1. The vertex table of the polytope index for Figure 1.

Id	Coords	Type	x-LE	x-RE	y-LE	y-RE
p_1	(1,3)	p	(h_1,p_1)	-	(h_2,p_1)	-
p_2	(2,1)	p	(h_2,p_2)	-	(v_2,p_2)	-
p_3	(3,5)	p	(h_3,p_3)	-	(h_4,p_3)	-
p_4	(5,2)	p	(v_5,p_4)	-	(v_4,p_4)	-
p_5	(4,7)	p	(h_5,p_5)	-	(h_7,p_5)	-
p_6	(7,4)	p	(v_7,p_6)	-	(v_6,p_6)	-
p_7	(6,6)	p	(h_7,p_7)	-	(v_7,p_7)	-
h_1	-	s	-	(h_1,p_1)	(O,h_1)	(h_1,h_3)
h_2	-	s	-	(h_2,p_2)	(v_1,h_2)	(h_2,p_1)
h_3	-	s	-	(h_3,p_3)	(h_1,h_3)	(h_3,h_5)
h_4	-	s	-	(h_4,v_5)	(v_3,v_4)	(h_4,p_3)
h_5	-	s	-	(h_5,p_5)	(h_3,h_5)	-
h_6	-	s	-	(h_6,v_7)	(v_5,h_6)	(h_6,h_7)
h_7	-	s	-	(h_7,p_7)	(h_6,h_7)	(h_7,p_5)
v_1	-	s	-	(O,v_1)	-	(v_1,h_2)
v_2	-	s	(v_1,v_2)	(v_2,v_3)	-	(v_2,p_2)
v_3	-	s	(v_2,v_3)	(v_3,v_4)	-	(v_3,h_4)
v_4	-	s	(v_3,v_4)	(v_4,v_6)	-	(v_4,p_4)
v_5	-	s	(h_4,v_5)	(v_5,p_4)	-	(v_5,h_6)
v_6	-	s	(v_4,h_6)	-	-	(v_6,p_6)
v_7	-	s	(h_6,v_7)	(v_7,p_6)	-	(v_7,p_7)

Table 2. The edge table of the polytope index for Figure 1.

Label	Type	lv	rv	LF	RF
(h_1,p_1)	p	h_1	p_1	F_1	F_3
(h_2,p_2)	p	h_2	p_2	F_2	F_3
(h_3,p_3)	p	h_3	p_3	F_3	F_5
(v_5,p_4)	p	v_5	p_4	F_4	F_6
(h_5,p_5)	p	h_5	p_5	F_5	X
(v_7,p_6)	p	v_7	p_6	F_6	X
(h_7,p_7)	p	h_7	v_7	F_7	X
(h_2,p_1)	p	h_2	p_1	F_1	F_3
(v_2,p_2)	p	v_2	p_2	F_2	F_3
(h_4,p_3)	p	h_4	p_3	F_3	F_5
(v_4,p_4)	p	v_4	p_4	F_4	F_6
(h_7,p_5)	p	h_7	p_5	F_5	X
(v_6,p_6)	p	v_6	p_6	F_6	X
(v_7,p_7)	p	v_7	p_7	F_7	X
(h_4,v_5)	s	h_4	v_5	F_4	F_5
(h_6,h_7)	s	h_6	h_7	F_5	F_7
(O,h_1)	s	O	h_1	X	F_1
(v_1,h_2)	s	v_1	h_2	F_1	F_2
(h_1,h_3)	s	h_1	h_3	X	F_3
(v_3,h_4)	s	v_3	h_4	F_3	F_4
(h_3,h_5)	s	h_3	h_5	X	F_5
(v_5,h_6)	s	v_5	h_6	F_5	F_6
(h_6,h_7)	s	h_6	h_7	F_5	F_7
(O,v_1)	s	O	v_1	X	F_1
(v_1,v_2)	s	v_1	v_2	X	F_2
(v_2,v_3)	s	v_2	v_3	X	F_3
(v_3,v_4)	s	v_3	v_4	X	F_4
(v_4,v_6)	s	v_4	v_6	X	F_6

Table 3. The face table of the polytope index for Figure 1.

F	x-LE	x-RE	y-LE	y-RE
F_1	(O,h_1)	(h_2,p_1)	(O,v_1)	(h_1,p_1)
F_2	(v_1,h_2)	(v_2,p_2)	(v_1,v_2)	(h_2,p_2)
F_3	(h_1,h_3)	(h_4,p_3)	(h_1,p_1)	(h_3,p_3)
F_4	(v_3,h_4)	(v_4,p_4)	(v_3,v_4)	(v_5,p_4)
F_4	(h_3,h_5)	(h_7,p_5)	(h_3,p_3)	(h_5,p_5)
F_6	(v_5,h_6)	(v_6,p_6)	(v_5,p_4)	(v_7,p_6)
F_7	(h_6,h_7)	(v_7,p_7)	(h_6,v_7)	(h_7,p_7)

Fast Piercing of Iso-Oriented Rectangles

Christos Makris Athanasios Tsakalidis

Department of Computer Engineering and Informatics, University of Patras 26500 Patras, Greece AND
Computer Technology Institute P.O. BOX 1122, 26110 Patras, Greece
e-mail address: makri@ceid.upatras.gr

Abstract

We present new algorithms for the problem of k-piercing of a set of n iso-oriented rectangles in the plane for $k \geq 4$. Our results are comprised of two $O(nlogn)$ time algorithms for $k = 4$ and $k = 5$, and an $O(n^{k-4} \log n)$ time algorithm for $k > 5$.

1 Introduction

Let S be a set of n convex objects in R^d. The set S is k-pierceable if there exists a set of k piercing points which intersect every member of S. The k-piercing problem is to determine whether S is k-pierceable and, if so, to produce a set of k piercing points. One important special case of the problem arises when we consider the set S as being comprised from a set of axis-parallel rectangles in the plane. The 1-, 2-, 3-piercing problems for a set of axis-parallel rectangles have been solved in linear time in [3,4]. For $k \geq 4$ the best known bounds were given in [7] where the k-piercing problem was solved in $O(n \log^{k-1} n)$ time and $O(n \times polylog(n))$ space for $4 \leq k < 6$, and in $O(n^{k-4} \log^5 n)$ time for $k \geq 6$. The results in [7] for $k > 4$ are based on the algorithmic technique used to solve the 4-piercing problem for a set of axis parallel rectangles. In a recent paper Michael Segal ([6]) using a different geometric transformation than that in [7] was able to solve the 4-piercing problem in $O(nlogn)$ time and the k-piercing problem for $k \geq 6$ in $O(n^{k-4} \log^4 n)$ time. With his approach he handles also the 1,2,3-problems but not the 5-problem.

In this paper we will show that by using the same geometric observations as in [7] but with more appropriate data-structuring techniques and some extra observations, we can obtain: (i) an $O(n)$ space, $O(nlogn)$ time algorithm for the 4-piercing problem, (ii) an $O(nlogn)$ space, $O(nlogn)$ time algorithm for the 5-piercing problem and (iii) an $O(n^{k-4} \log n)$ time algorithm for the k-piercing problem where $k \geq 6$, thus improving the previous solutions in terms of space, time bounds. The above improvements are based in the replacement of the sweep routine used in [7] by two independent processes (i) a preliminary sweep that produces a set of candidate k-tuples of points and (ii) a filtering algorithm that removes all the tuples that do not pierce the set of rectangles. As we will show for $k = 4, 5$ the above processes are based on dominance searching data structures in the $(k-2)$-dimensional space, and since 2- and 3-dimensional static dominance searching can be performed in $O(logn)$ query time we are able to achieve $O(nlogn)$time bounds.

Before closing this section we state briefly the basic points of the geometric observations in [7]. Let S be the set of rectangles we want to test for k-pierceability. Let L, R, T, B be the lines containing the leftmost right edge, the rightmost left edge, the highest bottom and the lowest top edge of the rectangles in S. Consider the closed left halfplane H^L bounded by L, the closed right halfplane H^R bounded by R, the closed top halfplane H^T bounded by T, and the closed bottom halfplane H^B bounded by B. Let $H\_^X$ denote the closure of the complement of H^X for $X = L, R, T, B$. Finally let $S_0 := \bigcap_{X=L,R,T,B} H\_^X$. The region S_0 (see fig. 1)) is called the location domain of S a term that is used in order to exhibit the relationship of S_0 with the location of the piercing points. If S_0 is empty then S is traversed by an axis parallel line. In this case it is known ([7,2]) that for any k the k-piercing problem can be solved in $O(n)$ time. Assume now that S_0 is not empty. The following lemma based on a careful geometric observation was stated in [7].

Lemma 1 *Assume that S is k-pierceable but has no axis parallel traversing line. Then there is a set of k piercing points in S_0 with each side of the boundary of S_0 containing one of the piercing points (the sides are considered as relatively closed, that is, a vertex of S_0 is contained in its two incident edges)*

2 4-piercing of rectangles

We are given a set $S = \{s_1, s_2, \ldots, s_n\}$ of rectangles and we want to find if there exists 4 points that stab S. First we construct in $O(n)$ time the location domain S_0 of the set S. If S_0 is empty then as mentioned before we can test for 4-pierceability in $O(n)$ time So

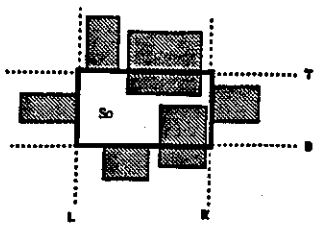


Figure 1:

assume that S_0 exists. We can decide whether there exists a piercing set of four points one of which is a vertex of S_0 by taking each such vertex and testing for 3-pierceability the rectangles disjoint from it in linear time. So we may assume that all the four piercing points lie on S_0 each lying in the relative interior of a distinct side. This case is handled as follows: Let t denote the intersection of the rectangle, whose bottom side lies on the line T of S_0, with the line T. Let $I_j = t \cap s_j$ for $j = 1, \ldots, n$, where $s_j (j = 1, \ldots, n)$ are the rectangles comprising the set S. The endpoints of the intervals I_j partition t into at most $2n-1$ "atomic" intervals. Our algorithm proceeds in two stages. In the first stage a sweep is performed moving on the atomic intervals from left to right and placing the top piercing point p_1 in each of them. For an atomic interval J the set S_J of rectangles of S that do not contain p_1 remains unchanged as p_1 varies within J. So we may assume that p_1 is the middle point of an atomic interval. When we pass from an interval J to an interval J' the set $S_{J'}$ differs from S_J by a single rectangle being added or removed. At each interval we produce two 4-tuples (p_1, p_2, p_3, p_4) of points that are candidates for piercing. So at the end of stage 1 we have a set C of $m = O(n)$ 4-tuples (p_1, p_2, p_3, p_4) to test if these tuples pierce the set S. By an observation on the type of restrictions these points obey we will show how this extraction can be done in $O(n \log n)$ time and $O(n)$ space. Note that stage 1 is similar to the sweep performed in [7] however here we don't test immediately if a candidate 4-tuple really pierces S. This saves us from the extra logarithmic factors in both time and space bounds. In stage 2 we examine if any of these 4-tuples is indeed a 4-piercing tuple for S. As we will show this problem can be solved in $O(n)$ space and $O(n \log n)$ time thus getting an overall improvement in both time and space bounds.

Stage 1

Let $s = [s_L : s_R] \times [s_B : s_T]$ be a rectangle of S. We map each rectangle $s \in S$ to 12 2-dimensional points (s_{x_i}, s_{y_i}) as follows:

$s_{x_1} = s_L$, $s_{y_1} = s_B$ $\quad s_{x_7} = s_B$, $s_{y_7} = s_R$
$s_{x_2} = s_R$, $s_{y_2} = s_B$ $\quad s_{x_8} = s_T$, $s_{y_8} = s_R$
$s_{x_3} = s_B$, $s_{y_3} = s_B$ $\quad s_{x_9} = s_L$, $s_{y_9} = s_L$
$s_{x_4} = s_T$, $s_{y_4} = s_B$ $\quad s_{x_{10}} = s_R$, $s_{y_{10}} = s_L$
$s_{x_5} = s_L$, $s_{y_5} = s_R$ $\quad s_{x_{11}} = s_R$, $s_{y_{11}} = s_L$
$s_{x_6} = s_R$, $s_{y_6} = s_R$ $\quad s_{x_{12}} = s_T$, $s_{y_{12}} = s_L$

(i) Construct the partition of the top interval t, as defined above, into up to $2n-1$ atomic intervals (time=$O(n \log n)$)

(ii) Sweep over these intervals from left to right. For each interval J, maintain the set S_J of rectangles of S not containing J into 12 priority search trees $PST_i (i = 1, \ldots, 12)$ where PST_i stores the points (s_{x_i}, s_{y_i}) as defined above. A priority search tree (PST) is a blend of a search tree and a heap that can store a set P of n 2-dimensional points (x_i, y_i). A description of this well known data structure can be found in [5]. It can be updated in $O(\log n)$ time per insertion and deletion and among the others permits the following query in $O(\log n)$ time:
$Miny\_PST(x_0 < x < x_1) = min\{y; \exists x; (x_0 < x < x_1) \wedge ((x, y) \in P)\}$ (if the heap is built on the inverse order of the y-axis then a PST can instead support the query $Maxy\_PST(x_0 < x < x_1)$). The choice of the heap order for each of the above PST's can be easily deduced from the next steps of our algorithm. Initialize the data structures with the set of rectangles not containing the left endpoint of t.

(iii) Let $S_0(J)$ be the location domain of S_J. It can be easily verified that the bottom, left and right sides of $S_0(J, v)$ lie on the same lines as the corresponding sides in S_0 and only the top side of $S_0(J)$ changes. Find the new top side by retrieving the x-coordinate of the last leaf of PST_3 and so construct the $S_0(J)$ (time=O(1)).

(iv) Our assumptions on the location of the piercing points, imply that one of the three piercing points of S_J must be one of the two top vertices of $S_0(J)$. We will place the second piercing point at each of these vertices. Let $v(v_x, v_y)$ be that vertex, and wlog assume that it is the top-left vertex of $S_0(J)$.

(v) Let $S(J, v)$ be the set of all rectangles from S_J that do not contain v. We need to find $S_0(J, v)$. It is easy to see that the bottom and right edges of $S_0(J, v)$ lie on the same lines as the corresponding sides in S_0. So we need to find the new left and top sides of $S_0(J, v)$. Since $S(J, v) := \{s \in S_J : (s_T < v_y) \vee (s_B > v_y) \vee (s_L > v_x) \vee (s_R < v_x)\}$ we have:
new
top side:=max$\{maxy_PST_1(x > v_x), maxy_PST_3(x < v_x), maxy_PST_5(x > v_y), maxy_PST_4(x < v_y)\}$
new
left side:=min$\{miny_PST_5(x > v_x), miny_PST_6(x < v_x), miny_PST_7(x > v_y), miny_PST_8(x < v_y)\}$
The above relations imply that the new $S_0(J, v)$ can be found in $O(\log n)$ time. From elementary geometric arguments it follows that the only candidate

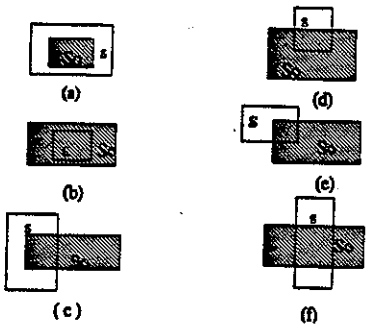


Figure 2:

points for the remaining two points are the top-right w and the bottom-left vertex z of $S_0(J,v)$. (*Remark:* if we took instead the rightmost vertex v' at step (iv) then we would have to find the new right and top sides of $S_0(J,v')$ and so we would have to query $PST_1, PST_2, PST_3, PST_4$ (as previously) and $PST_9, PST_{10}, PST_{11}, PST_{12}$ and the remaining two piercing points would be be the top left w' and the bottom right vertex z' of $S_0(J,v')$).

(vi) From the arguments of step (v) it follows that (middle point of J,v,w,z) and (middle point of J,v',w',z') are the two candidate 4-tuples of points contributed by J. We add these 4-tuples into a set C of candidate 4-tuples of points.

(vii) Move to the next atomic interval J' of t. Update the 12 PST's with the insertion or deletion of the rectangle by which S_J and $S_{J'}$ differ, and repeat the above steps to J'.

Since all the priority search trees can be updated and queried in $O(logn)$ time using $O(n)$ space we conclude that stage 1 can be completed in $O(nlogn)$ time and $O(n)$ space, producing a set C of $m = O(n)$ candidate 4-tuples of piercing points.

Stage 2

Let $C = \{c_1, c_2, \ldots, c_m\}$ be the set of candidate 4-tuples after the end of stage 1. We write a 4-tuple c_i as $c_i = (p_1^i, p_2^i, p_3^i, p_4^i)$ where p_1^i lies on the top side of S_0, p_2^i lies on the right side of S_0, p_3^i lies on the bottom side of S_0 and p_4^i lies on the left side of S_0. We also implement C as a doubly-threaded linear linked list. Stage 2 consists of the following steps:

(i) Store all the tuples in a constant number $A_1, A_2, \ldots, A_t$ of auxiliary structures where each structure has query/update time $O(logn)$ and uses linear space. The appearances of each tuple in $C, A_1, \ldots, A_t$ are linked together with bidirectional pointers,

(ii) For every rectangle $s_i \in S$ do:

1. using the structures $A_1, A_2, \ldots, A_t$ find the 4-tuples that do not pierce s_i.

2. delete all these tuples from $A_1, \ldots, A_t$ and C.

It is clear that if we have in our disposal these structures then steps (i) and (ii) can be performed in a total $O(nlogn)$ tine and linear space and the list C after the end of stage 2 will contain all the tuples that pierce S. If C is empty then S is not 4-piercable otherwise it is. So, the main obstacle in obtaining a solution is to find the appropriate data structures that will compute for each rectangle s in S the subset C_s of C consisting of the 4-tuples that that do not pierce s. The crucial observation that will help us is to look at the relationship that s, C have with the location domain S_0.

We have the following cases (see fig. 2):

(a) s completely covers S_0. In this case all the tuples of C pierce s,

(b) s is contained in S_0. In this case no tuple of C can pierce s and so S is not 4-pierceable.

(c) s fully contains an edge of S_0. In this case since every tuple of C has a point in one side of S_0 it follows that s is pierced by every 4-tuple of C.

(d) s cuts only an edge of S_0, and assume w.l.o.g that this is the top edge. Then we have:
$C_s = \{c_i \in C | (p_{1x}^i < s_L) \vee (p_{1x}^i > s_R)\}$.

(e) s cuts through two consecutive (in the clockwise order) edges of S_0, and assume w.l.o.g. that these edges are the top and the left of S_0. Then we have:
$C_s = \{c_i \in C | (p_{1x}^i > s_R) \wedge (p_{4y}^i < s_B)\}$

(f) s cuts through two opposite sides of S_0, and assume w.l.o.g. that these edges are the top and the bottom of So. Then we have:
$C_s = \{c_i \in C | ((p_{1x}^i > s_R) \wedge (p_{3x}^i > s_R)) \vee ((p_{1x}^i < s_L) \wedge (p_{3x}^i > s_R)) \vee ((p_{1x}^i > s_R) \wedge (p_{3x}^i < s_L)) \vee ((p_{1x}^i < s_L) \wedge (p_{3x}^i < s_L))\}$

As it is easy to see each of the above cases can be handled by a constant number of data structures each storing a set of 2-dimensional points and supporting queries of the form: *given a query point* (x_0, y_0) *find all the points (x,y) such that* $(x \oplus x_0) \wedge (y \oplus y_0)$, *where* $\oplus$ *any of* $<, >$. A priority search tree can support such operations in $O(logn)$ query and update time. So the solution to our problem is to enumerate all the possible relations of cases (d), (e), (f), (g) and for each such possible relation to create a PST that stores all the tuples of C (to be more precise case (d) can be handled by a simple binary balanced tree). Each tuple c_i appear in these trees as a two-dimensional point according to the above relations, and all the appearances of a tuple c_i in these trees and in C are linked together by bidirectional pointers. Since the number of possible relations is constant it follows that we need a constant number of PSTs. Since a PST can be updated in $O(logn)$ time and supports queries in $O(logn + k)$ time (k the output size) we get the following theorem:

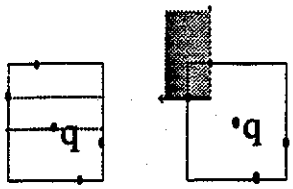


Figure 3:

Theorem 2. *The 4-piercing problem for n axis parallel rectangles can be solved in $O(n \log n)$ time and $O(n)$ space.*

3 5-piercing of rectangles

We are given a set $S = \{s_1, s_2, \ldots, s_n\}$ of rectangles and we want to find if there exists 5 points that stab S. First compute in $O(n)$ time the location domain S_0 of S. If S_0 is empty then we can test for 5-pierceability in $O(n)$ time. So assume that S_0 exists. We can easily decide whether there is a piercing set of five points one of which is a vertex of S_0 by taking each such vertex and testing for 4-pierceability the rectangles disjoint from it in $O(nlogn)$ time. So we may assume that we have four piercing points with each point lying in the relative interior of a distinct side and the fifth piercing points can lie anywhere in S_0 except the four corners. Let q be the fifth point. We can use a remark stated in [7] according to which at least one of the piercing points on the left and right sides of S_0 lie above q or at least one of these points lie below q. So suppose that at least one of these points lie above q and suppose that this point lies on the left side of S_0. Assume also that the point in the left side is higher (see fig. 3) than the point in the right side (the other constant number of cases can be handled in a similar way). Our algorithm again consists of two stages a stage 1 that collects a set of candidate 5-tuples and a stage 2 that filters out from these tuples those that do not pierce S.

The difference here is that stage 1 is broken in two phases with the first phase collecting a set C of $O(n)$ candidate 5-tuples each of which has only three defined points and a phase 2 which fills each of these candidate 5-tuples with the two remaining points. Phase 1 is completely analogous to the stage 1 algorithm for 4-piercing while phase 2 uses the geometric observations stated previously for the stage 2 of the 4-piercing algorithm. More precisely we have:

Phase 1:

(i) Obtain the partition of the top interval t of S_0, as defined in the previous section into up to $2n-1$ atomic intervals (time=$O(nlogn)$).

(ii) Sweep over these intervals from left to right. For each interval J, maintain the set S_J of rectangles of S not containing J into 8 priority search trees $PST_i (i = 1, \ldots, 8)$. These priority search trees are the same as in the 4-piercing algorithm. Initialize the data structures with the set of rectangles not containing the left endpoint of t.

(iii) Let $S_0(J)$ be the location domain of S_J. We observe again that the lines incident to the left, right and bottom sides of $S_0(J)$ are the same lines L, R, B defining S_0 and only the top side of $S_0(J)$ changes. Find the new top side by retrieving the x-coordinate of the last leaf of PST_3 and so construct the $S_0(J)$ (time=O(1)).

(iv) Our assumptions on the location of the piercing points imply, that the next piercing point must be the top left vertex of $S_0(J)$. Let $v(v_x, v_y)$ be that vertex.

(v) Let $S(J, v)$ be the set of all rectangles from S_J that do not contain v. We need to find $S_0(J, v)$. It is easily verified that the bottom and right sides of $S_0(J, v)$ lie on the same lines as the corresponding in S_0. So we need to find the new left and top sides of $S_0(J, v)$. Using the eight priority search trees we can find these sides in $O(\log n)$ time. Then according to our assumptions the third candidate piercing point must be either the top left vertex w_1 of $S_0(J, v)$, or the bottom left vertex w_2 of $S_0(J, v)$, or the top right vertex w_3 of $S_0(J, v)$. We add (middle point of J,v,w_1, $\star$, $\star$), (middle point of J,v,w_2,$\star$,$\star$), (middle point of J,v,w_3,$\star$,$\star$) in a set C of unfilled candidate piercing tuples. With each entry we also store the rectangle $S_0(J, v)$.

(vi) Move to the next atomic interval J' of t. Update the 8 PST's with the insertion or deletion of the rectangle by which S_J and $S_{J'}$ differ, and repeat the above steps to J'.

Phase 2: The phase 2 algorithm examines each of the tuples in C and tries to fill out the remaining two unfilled points. Let $c = (v_1, v_2, v_3, \star, \star)$ be an unfilled tuple where v_1 lies on the top side of S_0, v_2 lies on the left side of S_0 and v_3 can lie either on the right side of S_0, or the bottom side of S_0, or the interior of S_0. Assume w.l.o.g. that v_3 lies in the interior of S_0. With c we also have stored the location domain $S_0(J, v_2)$ constructed in step (v) before and we know that v_3 lies on the top left vertex of $S_0(J, v_2)$. Let $S(v_1, v_2, v_3)$ be the rectangles that are not pierced by any of v_1, v_2, v_3 and let $S_0(v_1, v_2, v_3)$ be their corresponding location domain. The crucial observation here is that if we find $S_0(v_1, v_2, v_3)$ then we have a constant number of cases where to put the two remaining points of c and so the tuple c will give rise to a constant number of candidate tuples with all their points filled. To find $S_0(v_1, v_2, v_3)$ we note that the bottom and right sides of it are the same as $S_0(J, v_2)$ so we only need to find the new left and top sides. We have:

(1) new top side:=$\max\{s_B : s \in S(v_1, v_2, v_3)\}$

(2) new left side:=min$\{s_R : s \in S(v_1, v_2, v_3)\}$
In order to answer the queries (1), (2) we will store the rectangles in S in a constant number of auxiliary structures, $A_1, A_2, \ldots, A_t$. These structures are built according to the relation the rectangles have with S_0 (as depicted in figure 2) and the specific combination of the location of v_1, v_2, v_3. Since the number of combinations of the locations of v_1, v_2, v_3 is constant and the number of the cases is constant we have a constant number of structures. So, let's examine again the cases of figure 2. We have:
(a) The rectangles in this case do not contribute to $S(v_1, v_2, v_3)$
(b) The rectangles in this case that are not pierced by v_1, v_2, v_3 are those that are not pierced by v_3.
(c) (i) The rectangles contain the top or left edge of S_0. The rectangles in this case do not contribute to $S(v_1, v_2, v_3)$ (they are pierced either by v_1 or by v_2)
(ii) The rectangles contain the right edge of S_0. The rectangles in this case that are not pierced by v_1, v_2, v_3 are those that are not pierced by v_1, v_3.
(iii)The rectangles contain the bottom edge of S_0. The rectangles in this case that are not pierced by v_1, v_2, v_3 are those that are not pierced by v_2, v_3.
(d) (i) The rectangles cut the top edge of S_0. The rectangles in this case that are not pierced by v_1, v_2, v_3 are those that are not pierced by v_1, v_3.
(ii) The rectangles cut the bottom edge of S_0. The rectangles in this case that are not pierced by v_1, v_2, v_3 are those that are not pierced by v_3.
(iii) The rectangles cut the right edge of S_0. The rectangles in this case that are not pierced by v_1, v_2, v_3 are those that are not pierced by v_3.
(iv) The rectangles cut the left edge of S_0. The rectangles in this case that are not pierced by v_1, v_2, v_3 are those that are not pierced by v_2, v_3.
(e) (i) The rectangles contain the top left vertex of S_0: In this case by construction every such rectangle is pierced either by v_1 or by v_2 (it is impossible to have rectangles contained in the shaded region of fig. 3)
(ii) The rectangles contain the top right vertex of S_0. The rectangles in this case that are not pierced by v_1, v_2, v_3 are those that are not pierced by v_1, v_3.
(iii) The rectangles contain the bottom left vertex of S_0. The rectangles of this case that are not pierced by v_1, v_2, v_3 are those that are not pierced by v_2, v_3.
(iv) The rectangles contain the bottom right vertex of S_0. The rectangles in this case that are not pierced by v_1, v_2, v_3 are those that are not pierced by v_3.
(f) (i) The rectangles cut through the top and bottom edge of S_0. The rectangles in this case that are not pierced by v_1, v_2, v_3 are those that are not pierced by v_1, v_3.
(ii) The rectangles cut through the left and right edge of S_0. The rectangles in this case that are not pierced by v_1, v_2, v_3 are those that are not pierced by v_2, v_3

So overall we conclude that for each case the rectangles that are not pierced by v_1, v_2, v_3 are those that are not pierced by at most two specific points of v_1, v_2, v_3.
Since for a rectangle $s \in S$ we have:
s not pierced by points p and q $\Leftrightarrow$
$\Leftrightarrow ((s_R < p_x) \vee (s_L > p_x) \vee (s_T < p_y) \vee (s_B > p_y)) \wedge ((s_R < q_x) \vee (s_L > q_x) \vee (s_T < q_y) \vee (s_B > q_y))$
and since $(A \vee B) \wedge C = (A \wedge C) \vee (B \wedge C)$, we have:
s not pierced by p,q $\Leftrightarrow \bigvee_{i=1,\ldots,8} term_i$, where $term_i = (s_{i_1} \otimes p_{i_1}) \wedge (s_{i_2} \otimes q_{i_2})$ and s_{i_1}, s_{i_2} can be any of s_L, s_R, s_T, s_B and p_{i_1} any of p_x, p_y and q_{i_2} any of q_x, q_y and $\otimes$ any of $<, >$.

From the above discussion it follows that in order to answer the questions (1), (2), for an unfilled tuple c, we need to preprocess the set of rectangles for each of the cases (a),(b),(c), (d), (e), (f) in a constant number of structures storing three dimensional points (x, y, z) and able to answer, for a given pair (x_1, x_2) queries of the form:
$findmin(max)\_z(x_1, x_2) = min(max)\{z|(x, y, z)$ corresponds to a rectangle, and $((x \otimes x_1) \wedge (y \otimes x_2))\}$.
Then we collect the minimum (or the maximum) of all the (constant number) of answers and we get the answer to (1), (2). Structures suitable for this kind of queries are proposed in [1] where it is shown how to preprocess a set of three dimensional points in $O(n \log n)$ time and space such that such queries can be answered in $O(\log n)$ time. These structures use the power of the RAM model of computation since they are based on computing least common ancestors in O(1) time. So we have shown that we can, in $O(nlogn)$ time and space, collect a set C of $m = O(n)$ 5-tuples of candidate piercing points.
Stage 2 Stage 2 uses again the same case observations as in stage 2 for the 4-piercing algorithm. However now we follow the reverse approach that is we store the set S in a constant number of data structures and test each tuple of C separately as follows:

1. Store S in a constant number of data structures $A_1, A_2, \ldots, A_t$.

2. Take each 5-tuple c_i of C and using $A_1, A_2, \ldots, A_t$ test if c_i pierces S.

3. If a tuple of C has been found to pierce S then S is 5-pierceable otherwise it is not.

Using the same observations as in the stage 2 of the 4-piercing algorithm we see that for each case the relation: a rectangle s in not pierced by a 5-tuple is equivalent to the relation that s in not pierced by at most three of the points in the tuple. By omitting easy but tedious details we just say that for each

class of rectangles (according to the specific cases of fig. 2) we create a constant number of data structures each storing 3-dimensional points (x,y,z) and being able to answer queries of the form: *given a triple* (x_1, x_2, x_3) *find if there is a point* (x, y, z) *such that* $(x \otimes x_1) \wedge (y \otimes x_2) \wedge (z \otimes x_3)$. Then given a specific 5-tuple c_i we accomplish steps (ii) and (iii) by querying in the appropriate way the appropriate data structures. If all the structures report "no" as an answer then c_i pierces S otherwise c_i does not pierce S. In [1] it has been shown how to build, in $O(n \log n)$ time and space, data structures that answer such queries in $O(\log n)$ time. So we conclude:

Theorem 2 *The 5-piercing problem for n axis parallel rectangles can be solved in* $O(n \log n)$ *time and* $O(n \log n)$ *space.*

Remark: In the full paper we show that by combining, previously published dominance solutions, with the approach used in [1] it is possible to reduce the space requirements of the structures we use from [1], to $O(n)$, thus reducing the space of the above theorem to linear (the time bounds are unaffected).

4 k-piercing of rectangles

Consider now the k-piercing problem for $k \geq 6$. The algorithm again is comprised by two stages, with the same functioning as before. The first stage can be implemented in a brute-force way to run in $O(n^{k-4} \log n)$ time by simply taking (as in [7]) all the possible combinations of points until we get ourselves in the case where we want 5 remaining points to fill in a candidate k-tuple, where we resort to the stage 1 of the 5-piercing algorithm. Note that the set C of candidate k-tuples built in the first stage has size $O(n^{k-4})$.

The stage 2 is analogous to the stage 2 of the 5-piercing algorithm. Here we will take advantage of the fact that the number of rectangles is $O(n)$ whereas the set C has quadratic or superquadratic size giving us the freedom to store S in structures of size $O(n^{1+\epsilon})$ where ϵ arbitrarily small positive constant. To be more specific the steps are:

1. Store S in a constant number of data structures $A_1, A_2, \ldots, A_t$.

2. Take each k-tuple c_i of C and using $A_1, A_2, \ldots, A_t$ test if c_i pierces S.

3. If a tuple of C has been found to pierce S then S is k-pierceable otherwise it is not.

It is obvious that a k-tuple $c = (p_1, p_2, \ldots, p_k)$ does not pierce S if the set of rectangles in S that are not pierced by c is not empty. Let V_c denote this set. We have that: $V_c = \{s \in S : V_1 \wedge V_2 \wedge V_3, \ldots, \wedge V_k\}$, where $V_i = (s_T < p_{i_y}) \vee (s_B > p_{i_y}) \vee (s_L > p_{i_x}) \vee (s_R < p_{i_x})$. Since $(A \vee B) \wedge C = (A \wedge C) \vee (B \wedge C)$ the set can be rewritten as: $V_c = \{s \in S : \bigvee_{i=1,\ldots,t} term_i\}$, where $term_i$ is the conjunction of one term of V_1, one term of V_2, one term of V_3 ,...,and one term of V_k and t is 4^k. It is easy to see that every $term_i$ is of the form: $term_i = \bigwedge_{j=1,\ldots,k} f_j \otimes t_j$, where f_j can be any of s_T, s_R, s_L, s_B and t_j can be any of p_{j_x}, p_{j_y} and $\otimes$ can be any of $<, >$. Each rectangle can give rise to $t = 4^k$ $(f_1, f_2, f_3, f_4, \ldots, f_k)$ points. So in order to find if c pierces S it suffices to store the rectangles of S in t structures $A_1, A_2, \ldots, A_t$ where structure A_i can deal with queries of the form proposed by $term_i$. Then we query each such structure with c and if all of these structures report output of zero size then c pierces S otherwise c does not pierce S. By using a multilevel range tree where each successive level has constant depth (as described in [8]), each A_i can be implemented to have size and preprocessing time $O(n^{1+\epsilon})$ where ϵ can be arbitrarily small so that each such query takes $O(\log n)$ time. So overall we have a time complexity of $max(O(n^{1+\epsilon}), O(n^{k-4} \log n)) = O(n^{k-4} \log n)$ and we conclude with the theorem:

Theorem 3 *The k-piercing problem for n axis parallel rectangles when* $k > 5$ *can be solved in* $O(n^{k-4} \log n)$ *time.*

References

[1] H. Gabow, J. Bentley, R. Tarjan, Scaling and related techniques for geometry problem, *Proc. 16th Annual Symp. On Theory of Comput.* pp. 135-143, 1984.

[2] M. Golumbic, Algorithmic Graph Theory, *Academic Press, New York,* 1980.

[3] M.T. Ko and Y.T. Ching, Linear time algorithms for the weighted tailored 2-partition problem and the weighted 2-center problem under $L\infty$ distance, *Discrete Applied Math.* 40, pp. 397-410, 1992

[4] M.T. Ko and R.C.T. Lee, On weighted rectilinear 2-center and 3-center problems, *Inform. Sci.* 54 (1991) 169-190.

[5] Kurt Mehlhorn, Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry, *Springer-Verlang,* 1984

[6] Michael Segal, On Piercing Sets of Axis-Parallel Rectangles, manuscript

[7] M. Sharir, E. Welzl, Rectilinear and Polygonal p-Piercing and p-Center Problems, *Proc 12th ACM Symp. on Computational Geometry,* pp.122-132, 1996

[8] M. van Kreveld, New results on Data Structures in Computational Geometry *Phd Thesis, Utrecht University,* 1992.

Shooter Location Problems Revisited [1]

Cao An Wang [2] and Binhai Zhu [3]

Abstract

In this paper, we propose an $O(n^5 \log n)$ time algorithm for the general *shooter location problem* thus solving an open problem posted by Nandy, Mukhopadhyaya and Bhattacharya. With geometric duality we also present a factor-2 approximation algorithm which runs in $O(n^5)$ time for this problem.

1 Introduction

Let $L = \{l_1, l_2, ..., l_n\}$ be a set of line segments with finite length in the plane. A shooter in this environment of L can fire or emit rays from a point in arbitrary directions such that a ray can penetrate all the line segments in its linear path of motion from its origin to infinite. *Shooter location problem* is to locate a position in the plane such that the number of shots fired from that position necessary to hit all the line segments in L is minimal over all other positions in the plane. This general problem is raised by Nandy et al. in [NMB96] as an open problem, they also solved a restricted case of shooter location problem; that is, the shooter is only allowed to move along a given line. They provided an $O(n^3)$ algorithm to find a position in the given line such that the shooter is able to fire the minimum number of shots necessary to hit all the line segments in L.

Shooter location problems can be regarded as a generalization of the well-known transversal or stabbing problem [Ede87]. In the environment of L, if one can locate a position at which a shooter is able to fire a single shot or two shots in the opposite directions to stab all the elements of L, then one has found a stabbing line. A natural generalized question is how to find a location where the shooter can fire a minimum number of shots necessary to hit all the elements in L.

A further inspection of the problem shows that this problem can be divided into a polynomial ($O(n^4)$) number of subproblems, and each can be solved in $O(n \log n)$ time. This gives us a polynomial $O(n^5 \log n)$ time solution. With geometric duality and $O(n^2)$ time and space preprocessing, a factor-2 approximation solution for each of the above subproblems can be obtained in $O(n)$ time. The efficiency of our method is based on an observation that the problem in the original form in the plane can be transformed into a new form in the dual space and the problem in new form can be solved more efficiently. This contributes an $O(n^5)$ time factor-2 approximate solution for the problem.

For some variations of this problem and some related references, readers are referred to [AB87], [BKKMSU95,HM90,HT92].

2 Preliminaries on Geometric Duality

We shall review a typical transformation from an orthogonal xy−coordinate system to another orthogonal $\alpha\beta$−coordinate system in the plane and state some properties with respect to the basic geometric

[1] This work is supported by NSERC grant OPG0041629 and a research grant from City University of Hong Kong.
[2] Department of Computer Science, Memorial University of Newfoundland, St.John's, NFLD, Canada A1B 3X8.
[3] Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong.

objects in these two coordinate systems.

Let p be a point in the plane. Point p can be represented in an orthogonal $xy-$coordinate system by its two coordinates (x_p, y_p). Let l be a line in the plane which is uniquely determined by a point and a slope in the $xy-$coordinate system. In more detail, let p with coordinates x_p and y_p be a point through which l passes and k_l be the real representing the slope of l. Then, every point (x, y) on l must satisfy the following formula

$$y - y_p = k_l * (x - x_p).$$

That is, $y = k_l * x + (y_p - k_l * x_p)$. Let $b_l = (y_p - k_l * x_p)$, we have that $y = k_l * x + b_l$. In general, a line can be represented as

$$y = k * x + b - - - -(1).$$

Consider another orthogonal coordinate system with α and β as variables (axes). We define a mapping $\mathcal{M} : k \to \alpha, b \to \beta$. Then, under the mapping $\mathcal{M}$, a line l with slope k_l and point at the $x-$axis b_l in the $xy-$coordinate system corresponds to a point (k_l, b_l) in the $\alpha\beta-$coordinate system.

Furthermore, a point p in the $xy-$coordinate system can be regarded as the intersection point of all the lines passing p in the plane. Thus, p can be represented as an infinite number of lines satisfying the following formula

$$y = k * x + (y_p - k * x_p)$$

for variable k in $[-\infty, \infty]$. Then, under the mapping $\mathcal{M}$, point p becomes a line in the $\alpha\beta-$coordinate system satisfying the following formula

$$\beta = -x_p * \alpha + y_p - - - -(2).$$

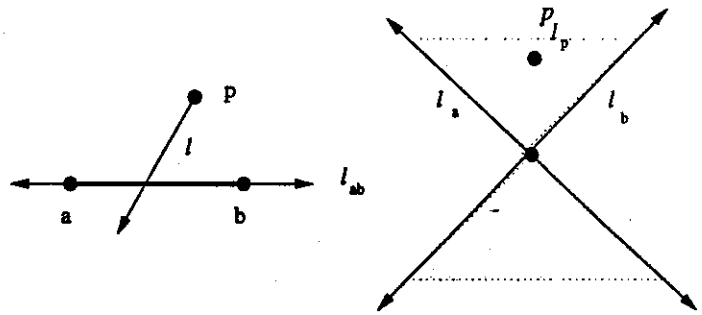


Figure 1: The left-hand side is a line segment in the plane with a ray (line) l crossing it from point p and the right-hand side is the double-wedge in the dual plane with point p_{l_p}.

The following properties are derivable. For convenient, we call the $xy-$coordinate system, the plane (or the primal space) and the $\alpha\beta-$coordinate system, the dual plane.

Property 1: (Refer to Figure 1.) A line segment $\overline{ab}$ in the plane corresponds to a *double-wedge* area bounded by the two lines l_a and l_b in the dual plane, where l_a and l_b correspond to a and b, respectively. The line containing $\overline{ab}$ in the plane, l_{ab}, corresponds to the tip point of the double-wedge.

A line crossing $\overline{ab}$ in the plane becomes a point in the double-wedge area.

Conversely, let $\mathcal{M}^{\nabla}$ be the mapping: $\alpha \rightarrow k, \beta \rightarrow b$. Then under $\mathcal{M}^{\nabla}$, a point $q(\alpha_q, \beta_q)$ in the dual plane becomes a line l_q satisfying $y = \alpha_q * x + \beta_q$ in the plane, and a line satisfying $\beta = -k_0 * \alpha + b_0$ in the dual plane corresponds a point (k_0, b_0) in the plane.

Let L be the given n line segments (objects). By Property 1, the endpoints of each l of L determines two lines in the dual plane. Thus, L determines at most $2n$ lines, denoted by L', in the dual plane. These lines L' can be organized into a data structure $A(L')$, the arrangement of L', in $O(n^2)$ time and $A(L')$ contains $O(n^2)$ cells [Ede87]. Each cell of $A(L')$ is the intersection area of certain double-wedges. We say these double-wedges are 'containing' the cell.

Property 2: Each point in a cell of $A(L')$ in the dual plane represents a line crossing the objects in L in the plane, where the objects correspond the double-wedges containing the cell.

3 Two solutions for the fixed shooter location problem

In this section, we present two solutions, one exact and one approximate, for the fixed shooter location problem.

3.1 Exact primal solution

In this subsection, we mention using the minimum clique cover in a circular-arc graph to solve a special case of our general problem. The problem, *fixed shooter location*, can be stated as follows. Given a point p in the environment of a set of n line segments L (objects) in the plane, find the minimum number of shots from p to hit all objects of L.

Our method works on the problem in the primal plane. We draw a big circle C centered at p which contains all the segments. For each segment $\overline{s_1 s_2} (\in L)$, we map it into an arc in the boundary of C by drawing rays from p and passing s_1 and s_2. All the mapped arcs form a *circular-arc graph* in which each node corresponds to an arc and an edge exists between two nodes if and only if their corresponding arcs overlap (intersect). We call this graph, denoted by $G(p)$, the *circular-arc graph induced from p*. The problem of finding the minimum number of shots from p necessary to hit all objects of L becomes that of computing the *minimum clique cover* of this circular-arc graph $G(p)$.

Although the problem of computing the minimum clique cover of a general graph is NP-Complete [GJ79], the problem of a circular-arc graph with n arcs can be solved in $O(n \log n)$ time [HT91] (in fact, after sorting the vertices of the arcs according to the polar angles around the center of the circle, the remaining steps take only $O(n)$ time.) Therefore we have that

Observation 1. The fixed shooter location problem can be solved in $O(n \log n)$ time in the primal space.

3.2 Approximate dual solution

In this subsection, we propose a factor-2 approximate solution for the fixed shooter location problem. We first construct the dual of all the segments l_i's in L and the dual of point p. The dual of each line segment is a double-wedge and the dual of p, denoted by l_p, is a line. Then we construct the arrangement of all the double-wedges $A(L')$. The following properties are easy to obtain. The intersection of

a double-wedge and l_p determines an interval in l_p. (This interval is finite if it is the intersection of one wedge of the double-wedge with l_p and the interval would be infinite if it is the intersection of the both wedges with l_p. In the latter case the interval can be regarded as connected in infinite if we think of l_p as a circle with an infinite length of radius.) There are $O(n)$ such intervals on l_p with respect to the arrangement of these double-wedges. The endpoints of these intervals are sorted once the arrangement for $A(L')$ is known. Thus, we can find the minimum clique cover of all the intervals in $O(n)$ time by the greedy algorithm proposed in [HT91]. Let K denote this minimum cover, and let T_K denote the dual (lines) of K in the primal space. Then T_K are the lines through p with the minimum size hitting all the objects in L. Since we only need rays at p not the lines through p, we simply divide each line in T_K at p into two rays and denote these rays by T_p. Thus, the size of T_p is twice as the size of T_K.

To see the above solution is a factor-2 approximate solution, note that if all the rays in the optimal solution are extended into lines, then the number of the extending lines must be equal to the number of lines in T_K. Otherwise, there would exist a solution with less rays or less lines, the former contradicts to the optimal solution of the problem and the latter contradicts to T_K is the minimum size of lines hitting L. The following is the detailed description of our approximate solution.

ALGORITHM FixedShooter(L,p)

Input: L (a set of n line segments with finite length), and p (the given point).
Output: T_p (the set of rays at p hitting all elements in L).
Method:

1. Find the dual of L, L', in the dual plane under $\mathcal{M}$ using formula (2).
2. Find $A(L')$ by the method in [Ede87].
3. Find the dual of p, l_p, in the dual plane under $\mathcal{M}$ using formula (2).
4. Walk on $A(L')$ along l_p to obtain the interval graph $G(l_p)$.
5. Find the minimal clique cover K of $G(l_p)$ using the method in [HT91].
6. Find the dual of this clique cover T_K in the primal space under $\mathcal{M}^{\nabla}$ using formula (1).
7. Divide each line in T_K into two rays starting at p, this is a set of rays T_p starting at p.

Lemma 1 *With geometric duality, a factor-2 approximate solution for the fixed shooter location problem can be obtained in $O(n^2)$ time (or equivalently, after $O(n^2)$ time and space preprocessing on constructing the arrangement of the double-wedges, in $O(n)$ time) by Algorithm* **FixedShooter**.

Proof As described above. □

4 Algorithm for the Shooter Location Problem

Let S be the endpoint set of L. Let $\mathcal{L}$ be the set of lines each of which is determined by a pair of points in S. Clearly, $\mathcal{L}$ has at most $O(n^2)$ lines. Let $A(\mathcal{L})$ be the arrangement of $\mathcal{L}$. Then, we have the following property.

Lemma 2 *In the primal space, the circular-arc graphs induced from points u and v are the same if u and v belong to the same cell in $A(\mathcal{L})$.*

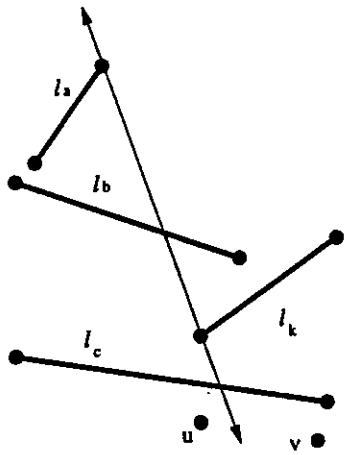


Figure 2: Illustration for the proof of Lemma 2.

Proof Note that in the optimal solution of the fixed shooter location problem, a ray in the solution can always be rotated around p, maintaining the same intersection relationship with the object line segments, until the ray touches an element (an endpoint) in S. Thus, after the above rotation the corresponding graph $G(p)$ remains the same. We call such a rotated ray, an *extremal* ray.

Let $G(u)$ and $G(v)$ be the circular-arc graphs induced from u and v, respectively. Assume that $G(u)$ and $G(v)$ are different, i.e. there exists at least one pair of objects, say l_a and l_k, such that an extremal ray from v crosses both l_a and l_k while an extremal ray from u misses one of l_a and l_k. Then, there must exist a line passing an endpoint of l_k and an endpoint of l_a which separates u and v. (Refer to Figure 2 for the claim.) This results in that u and v lie in different cells, a contradiction. □

Lemma 2 implies that for any two points u and v in the same cell of $A(\mathcal{L})$, the optimal solutions for the fixed shooter location problems, produced by the minimum clique cover algorithm of [HT91], will have the same size because the two induced graphs $G(u)$ and $G(v)$ are the same. (Note that the two solutions could be different but with the same size.)

There are at most $O(n^4)$ cells in $A(\mathcal{L})$. For each cell of $A(\mathcal{L})$, we choose a point and find the minimum set of shots at that point using **FixShooter**. We have thus reduced the problem into $O(n^4)$ *fixed shooter location* problems. Then, we compare the sizes of all these sets to obtain a set with the minimum size. Using the subroutine described in Observation 1, we obtain an exact solution in $O(n^5 \log n)$ time and using the subroutine described in Lemma 1, we obtain a factor-2 approximate solution in $O(n^5)$ time. (Note that in the latter case, the $O(n^2)$ time and space preprocessing for constructing $A(L')$ is done exactly once.) We have the following main result.

Theorem 1 *Given an environment L consisting of n line segments, the minimum set of shots (rays) at any point in the plane intersecting all elements in L can be computed in $O(n^5 \log n)$ time. A factor-2 approximate solution can be obtained in $O(n^5)$ time.*

5 Conclusion

In this paper, we present polynomial time exact and approximate algorithms for the shooter location problems. We have thus solved an open problem posted by Nandy et al. Whether or not our method can be extended into three or higher dimensional spaces to obtain a good approximate algorithm

remains a further work. When the location p is restricted to be on a line in the plane Nandy et al. provided an $O(n^3)$ time algorithm [NMB96]. It is also not clear whether this bound can be improved or not.

References

[AB87] M. Attallah and C. Bajaj, "Efficient algorithms for common transversal", *Information Processing Letters*, Vol. 25 (1987), pp. 87-91.

[BKKMSU95] F. Bauernoppel, E. Kranakis, D. Krizanc, A. Maheswari, I. Sack and J. Urrutia, "Optimal shooting: characterization and applications", *Proc. 22nd ICALP*, LNCS V.944, 1995, pp.220-231.

[Ede87] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1987.

[GJ79] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, 1979.

[HM90] R. Hassin and N. Megiddo, "Approximation algorithms for hitting objects with straight lines", *Discrete Applied Mathematics*, Vol.30 (1990), pp. 29-42.

[HT91] W. Hsu and K. Tsai, "Linear time algorithms on circular-arc graphs", *Information Processing Letters*, Vol. 40, (1991) pp.123-129.

[HT92] M. Holmeyer and S. Teller, "Stabbing isothetic boxes and rectangles in $O(n \log n)$ time", *Computational Geometry: Theory and Applications*, Vol.2, 1992, pp.201-207.

[NMB96] S. Nandy, K. Mukhopadhyaya and B. Bhattacharya, "Shooter location problems", *Proc. 8th Canadian Conference on Computational Geometry*, 1996, pp. 93-97.

Approximation Algorithms for Geometric Optimization Problems

(Extended Abstract)

Joseph S. B. Mitchell*

1 Introduction

Many network optimization problems in graphs are known to be hard to solve exactly (see Garey and Johnson [6]), and many of these remain hard for geometric instances of the same problem: the traveling salesperson problem (TSP), Steiner tree problem, k-MST problem, etc., are all known to be NP-hard even for point sets in the Euclidean plane.

An increasingly popular approach to "solving" NP-hard optimization problems is to obtain provably-good approximation algorithms, which are guaranteed, in polynomial time, to produce an answer that is close to optimal – say, whose objective function value at most some factor $c > 1$ times optimal, for a minimization problem. (Such an approximation algorithm is then called a "c-approximation" algorithm.)

A *polynomial time approximation scheme* (PTAS) is a method that allows one to compute a $(1+\epsilon)$-approximation to the optimal (minimum), in time that is polynomial in n, for any fixed $\epsilon > 0$. (In general, the dependence on ϵ may be exponential in $(1/\epsilon)$.)

The recent book editted by Hochbaum ([7]) contains several articles surveying the state of knowledge on approximation algorithms for NP-hard problems. In particular, the survey of Bern and Eppstein [3] gives an excellent overview of the subject of approximating NP-hard geometric optimization problems.

Approximation algorithms can also be quite useful for problems that are not necessarily NP-hard. First, the approximation algorithm may be considerably simpler and easier to implement than an algorithm that solves the problem to optimality. Further, the running time (both worst-case and average-case) for the approximation algorithm may be much better than that which is known for the exact solution, even when the exact algorithm has polynomial running time. Examples from geometry include various shortest path ([14]) and matching problems.

Further, approximation algorithms are known for some problems whose complexity status is still open, such as the minimum-weight triangulation problem.

In this talk, I will briefly survey some recent progress on approximation algorithms for geometric network optimization, and will spend some time detailing the method of m-guillotine subdivisions, which leads to PTAS's for several problems, including the geometric TSP, Steiner tree, and k-MST problems.

In the remainder of this extended abstract, I enclose the text from a recent note ([13]) on an improved PTAS based on a variant of m-guillotine subdivisions. The note refers to earlier work, which is available on the web. Also, as these results are constantly changing and improving, I refer the reader to my web page (or to personal email), for further updates.

2 A PTAS for Geometric Network Optimization

In this note, we show how a modification to our earlier results on guillotine subdivisions leads to an $n^{O(1)}$ time (deterministic) PTAS for Euclidean versions of various geometric network optimization problems on a set of n points in the plane. This improves the previous $n^{O(1/\epsilon)}$ time algorithms of Arora [1] and Mitchell [12].

Our methods are based on the concept of an "m-guillotine subdivision", which were introduced by Mitchell [11, 12]. Roughly speaking, an "m-guillotine subdivision" is a polygonal subdivision with the property that there exists a line ("cut"), whose intersection with the subdivision edges consists of a small number ($O(m)$) of connected components, and the subdivisions on either side of the line are also m-guillotine. The upper bound on the number of connected components allows one to apply dynamic programming to optimize over m-guillotine subdivisions, as there is a succinct specification of how subproblems interact across a cut.

Key to our method is a theorem showing that any polygonal subdivision can be converted into an appropriate m-guillotine subdivision by adding a set of edges whose total length is small: at most $\frac{c}{m}$ times that of the original subdivision (where $c = 1, \sqrt{2}$, depending on the metric). Key to our improvement over previous results on approximating with guillotine subdivisions is the notion of a "grid-rounded" m-guillotine subdivision, in which each connected component is also required to contain one of a small number of regularly spaced grid points. (These notions are made precise in the next section.) Then, exactly as in [12], we use dynamic programming to optimize over an appropriate class of m-guillotine subdivisions, resulting in, for any fixed m, $(1+\frac{c}{m})$-approximation algorithms that run in polynomial-time ($O(n^{O(1)})$), for various network optimization problems.

Related Work. There has been an abundance of work on the problems studied here, both on instances of the problems in graphs and on geometric instances. We refer the

*jsbm@ams.sunysb.edu, http://www.ams.sunysb.edu/~jsbm/, Dept. of Applied Mathematics and Statistics, State University of New York, Stony Brook, NY 11794-3600. Supported in part by Hughes Research Laboratories and NSF Grant CCR-9504192.

reader to some standard textbooks, such as [5, 8, 16]. For the particular problem of the TSP, there is a survey book edited by Lawler et al. [9], and for results on approximation theory and algorithms, there is the recent book edited by Hochbaum [7].

While the geometric optimization problems considered here are known to be NP-hard, polynomial-time approximation algorithms have been known previously that get within a constant factor of optimal. Further, polynomial-time approximation schemes were discovered last year, by Arora [1] and by Mitchell [11, 12].

This paper represents a continuation of our previous work on guillotine subdivisions ([11, 12, 15]), which in turn is based on the concept of "division trees" introduced by Blum, Chalasani, and Vempala [4, 15], and the guillotine *rectangular* subdivision methods of Mata and Mitchell [10]. Here, we obtain substantially better time bounds than before, improving the previous running time from $n^{O(m)}$ to $O(n^{O(1)})$.

Arora [2] has recently obtained a randomized algorithm whose expected running time is better than the deterministic time bound obtained here: He obtains a randomized algorithm with expected running time $O(n \log^{O(1/\epsilon)} n)$.

3 Grid-Rounded m-Guillotine Subdivisions

Definitions

We follow most of the notation of [11, 12]. We consider a polygonal subdivision ("planar straight-line graph") S that has n edges (and hence $O(n)$ vertices and facets). Let E denote the union of the edge segments of S, and let V denote the vertices of S. We can assume (without loss of generality) that S is restricted to the unit square, B (i.e., $E \subset int(B)$). Then, each facet (2-face) is a bounded polygon, possibly with holes. The *length* of S is the sum of the lengths of the edges of S. Assume, without loss of generality, that no two vertices of S have a common x- or y-coordinate.

A closed, axis-aligned rectangle W is a *window* if $W \subseteq B$. In the following definitions, we fix attention on a given window, W. We let $\overline{W} \subseteq W$ denote the minimal bounding box (axis-aligned rectangle) containing the vertices $V \cap W$ within W. Note that there are only $O(n^4)$ different possibilities for $\overline{W}$.

A line ℓ is a *cut* for E (with respect to W) if $\ell \cap int(W) \neq \emptyset$. The intersection, $\ell \cap (E \cap int(W))$, of a cut ℓ with $E \cap int(W)$ (the restriction of E to the window W) consists of a discrete (possibly empty) set of subsegments of ℓ. (Some of these "segments" may be single points, where ℓ crosses an edge.) The endpoints of these subsegments are called the *endpoints along* ℓ (with respect to W). (The two points where ℓ crosses the boundary of W are not considered to be endpoints along ℓ.) Let ξ be the number of endpoints along ℓ, and let the points be denoted by $p_1, \ldots, p_\xi$, in order along ℓ.

For a positive integer m, we define the *m-span*, $\sigma_m(\ell)$, of ℓ (with respect to W) as follows. If $\xi \leq 2(m-1)$, then $\sigma_m(\ell) = \emptyset$; otherwise, $\sigma_m(\ell)$ is defined to be the (possibly zero-length) line segment, $p_m p_{\xi-m+1}$, joining the mth endpoint, p_m, with the mth-from-the-last endpoints, $p_{\xi-m+1}$.

Given a line segment $\sigma = pq$ ($p \neq q$) and a positive integer M, consider the set of subsegments obtained by cutting pq into M equal-length segments; we define the *M-grid* of $\sigma = pq$ to be the set of $M+1$ endpoints of these subsegments. (In particular, the M-grid contains the two points p and q.)

A line ℓ is an *(m, M)-perfect cut with respect to W* if $\sigma_m(\ell) \subseteq E$, *and* each connected component of $\ell \cap E$ contains an M-grid point of the 1-span, $\sigma_1(\ell)$. In particular, if $\xi \leq 2(m-1)$, then ℓ is trivially an (m, M)-perfect cut (since $\sigma_m(\ell) = \emptyset$). Similarly, if $\xi = 2m-1$, then ℓ is m-perfect (since $\sigma_m(\ell)$ is a single point). Otherwise, if ℓ is m-perfect, and $\xi \geq 2m$, then $\xi = 2m$.

In the remainder of this paper, we fix $M = m(m-1)$ and assume that $m \geq 2$.

Finally, we say that S is a *grid-rounded m-guillotine subdivision with respect to window W* if either (1) $V \cap int(W) = \emptyset$; or (2) there exists an (m, M)-perfect cut, ℓ, with respect to the minimal window, $\overline{W} \subseteq W$, such that S is grid-rounded m-guillotine with respect to windows $W \cap H^+$ *and* $W \cap H^-$, where H^+, H^- are the closed halfplanes induced by ℓ. (Note that, since $\overline{W}$ is minimal, necessarily windows $W \cap H^+$ and $W \cap H^-$ will each contain a set of vertices distinct from that of W.) We say that S is a *grid-rounded m-guillotine subdivision* if S is grid-rounded m-guillotine with respect to the unit square, B.

The Approximation Theorem

The theorem below shows that grid-rounded m-guillotine subdivisions can approximate arbitrary subdivisions arbitrarily closely (as a function of m). Its proof directly follows that of [11, 12], with relatively minor changes to incorporate the concept of (m, M)-perfect cuts, which allow us to strengthen the requirements from that of m-perfect cuts to include the effect of rounding to the M-grid of the 1-span.

Theorem 1 *Let S be a polygonal subdivision, with edge set E, of length L. Then, for any positive integer m, there exists a grid-rounded m-guillotine subdivision, S_G, of length at most $(1 + \frac{2\sqrt{2}}{m})L$ whose edge set, E_G, contains E.*

Proof. We will convert S into a grid-rounded m-guillotine subdivision S_G by adding to E a new set of horizontal/vertical edges whose total length is at most $\frac{2\sqrt{2}}{m}L$. The construction is recursive: at each stage, we show that there exists a cut, ℓ, with respect to the current window W (which initially is the box B), such that we can afford to add the following set of segments to E:

- *("red" segment)* the m-span, $\sigma_m(\ell)$; and
- *("blue" segments)* a line segment on ℓ connecting each of the endpoints of $\ell \cap (E \cup \sigma_m(\ell))$ to a point of the M-grid of $\sigma_1(\ell)$.

By construction, once we add these segments to E, ℓ becomes an (m, M)-perfect cut with respect to W. The sense in which we can "afford" to add these segments is that we can charge off the lengths of the constructed segments to a portion of the length of the original edge set, E.

First, note that if an (m, M)-perfect cut (with respect to W) exists, then we can simply use it, and proceed, recursively, on each side of the cut. Thus, we assume that no (m, M)-perfect cut exists with respect to a given window, W.

We say that a point p on a cut ℓ is *m-dark with respect to ℓ and W* if, along $\ell^\perp \cap int(W)$, there are at least m endpoints (strictly) on each side of p, where $\ell^\perp$ is the line through p and perpendicular to ℓ.[1] We say that a subsegment of ℓ is

[1] We can think of the edges E as being "walls" that are not very effective at blocking light — light can go through $m-1$

m-dark (with respect to W) if all points of the segment are m-dark with respect to ℓ and W.

The important property of m-dark points along ℓ is the following: Assume, without loss of generality, that ℓ is horizontal. Then, if all points on subsegment pq of ℓ are m-dark, then we can charge the length of pq off to the bottoms of the first m subsegments, $E^+ \subseteq E$, of edges that lie above pq, and the tops of the first m subsegments, $E^- \subseteq E$, of edges that lie below pq (since we know that there are at least m edges "blocking" pq from the top/bottom of W). We charge pq's length half to E^+ (charging each of the m levels of E^+ *from below*, with $\frac{1}{2m}$ units of charge) and half to E^- (charging each of the m levels of E^- *from above*, with $\frac{1}{2m}$ units of charge). We refer to this type of charge as the "red" charge.

In addition to charging off the length of the m-dark portion of ℓ, in order to round to the M-grid of $\sigma_1(\ell)$, we are also going to charge off $(1/m)$th of the 1-dark portion of ℓ: If pq is 1-dark, then we charge $(1/m)$th of pq's length, by charging half of this length (i.e., $(1/2m)$th of the length of pq) off to the level of E that lies above pq, and half of it to the level of E that lies below pq. We refer to this type of charge as "blue" charge.

The *chargeable length* of a cut ℓ is defined to be the length of the m-dark portion of ℓ, plus $(1/m)$ times the length of the 1-dark portion of ℓ.

The *cost* of a cut, ℓ, is defined to be the length of the segments we must add to make the cut (m, M)-perfect. Thus, the cost of a cut ℓ is at most the length, $|\sigma_m(\ell)|$, of the m-span "red" segment, $\sigma_m(\ell)$, plus the lengths of the "blue" segments on ℓ connecting each of the endpoints of $\ell \cap (E \cup \sigma_m(\ell))$ to a point of the M-grid of $\sigma_1(\ell)$. Since there are at most $2m$ endpoints of $\ell \cap (E \cup \sigma_m(\ell))$, and two of these (the endpoints of $\sigma_1(\ell)$) are already at M-grid points of $\sigma_1(\ell)$, the total number of blue segments is at most $2m-2$. Further, each blue segment is at most one half of $\frac{|\sigma_1(\ell)|}{M}$, where $|\sigma_1(\ell)|$ is the length of the 1-span of ℓ. Thus, the overall cost of a cut ℓ is at most

$$|\sigma_m(\ell)| + (2m-2) \cdot \frac{1}{2} \cdot \frac{|\sigma_1(\ell)|}{M} = |\sigma_m(\ell)| + \frac{1}{m}|\sigma_1(\ell)|,$$

where we have used our choice o $M = m(m-1)$.

We call a cut ℓ *favorable* if the chargeable length of $\ell \cap W$ is at least as long as the cost of the cut.

The lemma below shows that a favorable cut always exists. For a favorable cut ℓ, we add its m-span to the edge set (charging off its length, as above), and recurse on each side of the cut, in the two new windows. After a portion of E has been charged *red* on one side, due to a cut ℓ, it will be within m levels of the boundary of the windows on either side of ℓ, and, hence, within m levels of the boundary of any future windows, found deeper in the recursion, that contain the portion. Thus, *no portion of E will ever be charged* red *more than once from each side, in each of the two directions (horizontal/vertical)*, so no portion of E will ever pay more than *twice* its total length, times $1/m$, in red charge ($\frac{1}{2m}$ from each side, for each of the two directions). Similarly, no portion of E will ever be charged *blue* more than once from each side, in each of the two directions, and when it is charged blue, it is charged at the rate of only $1/2m$ per unit length (per side, per direction); thus, no portion of E will ever pay more than its total length, times $2/m$, in blue charge.

walls, but is stopped when it hits the mth wall; then, p on a line ℓ is m-dark if p is not illuminated when light is shone in from the boundary of W, along the direction of $\ell^\perp$.

So far, this charging scheme gives rise to a total charge of at most $\frac{4}{m}L$. This factor can be improved slightly by noting that each side of an inclined segment of E may be charged red (resp., blue) *twice*, once vertically and once horizontally, so the red (resp., blue) charge assigned to a segment is at most $\frac{1}{m}$ times the sum of the lengths of its x- and y-projections, i.e., at most $\frac{\sqrt{2}}{m}$ times its length. This gives the overall charge of $\frac{2\sqrt{2}}{m}L$, as claimed.

It is also important to note that we are always charging portions of the original edges set E: The new edges added are never themselves charged, since they lie on window boundaries and cannot therefore serve to make a portion of some future cut m-dark or 1-dark.

(Note too that, in order for a cut ℓ to be favorable, but not (m, M)-perfect, there must be at least one vertex of V in each of the two open halfplanes induced by ℓ; thus, the recursion must terminate in a finite number of steps.) □

We now prove the lemma that guarantees the existence of a favorable cut. The proof of the lemma uses a particularly simple argument, based on elementary calculus (reversing the order of integration). It is based on the similar lemma that appears already in [11, 12], but we include its details here for completeness:

Lemma 2 *For any subdivision S, and any window W, there is a favorable cut.*

Proof. We show that there must be a favorable cut that is either horizontal or vertical.

Let $f(x)$ denote the cost of the vertical line, ℓ_x, through x; then,

$$f(x) = |\sigma_m(\ell_x)| + \frac{1}{m}|\sigma_1(\ell_x)|.$$

Then, $A_x = \int_0^1 f(x)dx$ is simply the area, $A_x^{(m)} = \int_0^1 |\sigma_m(\ell_x)|dx$, of the ($x$-monotone) region $R_x^{(m)}$ of points of B that are m-dark with respect to horizontal cuts, plus $(1/m)$ times the area, $A_x^{(1)} = \int_0^1 |\sigma_1(\ell_x)|dx$, of the ($x$-monotone) region $R_x^{(1)}$ of points of B that are 1-dark with respect to horizontal cuts. Similarly, define $g(y)$ to be the cost of the horizontal line through y, and let $A_y = \int_0^1 g(y)dy$.

Assume, without loss of generality, that $A_x \geq A_y$. We claim that there exists a horizontal favorable cut; i.e., we claim that there exists a horizontal cut, ℓ, such that its chargeable length (i.e., length of its m-dark portion, plus $(1/m)$ times the length of its 1-dark portion) is at least as large as the cost of ℓ ($|\sigma_m(\ell)| + \frac{1}{m}|\sigma_1(\ell)|$). To see this, note that A_x can be computed by switching the order of integration, "slicing" the regions $R_x^{(m)}$ and $R_x^{(1)}$ horizontally, rather than vertically; i.e., $A_x = \int_0^1 h(y)dy = \int_0^1 h_m(y)dy + \frac{1}{m}\int_0^1 h_1(y)dy$, where $h(y)$ is the chargeable length of the horizontal line through y, and $h^{(i)}(y)$ is the length of the intersection of $R_x^{(i)}$ with a horizontal line through y. (i.e., $h^{(m)}(y)$ (resp., $h^{(1)}(y)$) is the length of the m-dark (resp., 1-dark) portion of the horizontal line through y.) Thus, since $A_x \geq A_y$, we get that $\int_0^1 h(y)dy \geq \int_0^1 g(y)dy \geq 0$. Thus, it cannot be that for all values of $y \in [0,1]$, $h(y) < g(y)$, so there exists a $y = y^*$ for which $h(y^*) \geq g(y^*)$. The horizontal line through this y^* is a cut satisfying the claim of the lemma. (If, instead, we had $A_x \leq A_y$, then we would get a *vertical* cut satisfying the claim.) □

Algorithms

The dynamic programming algorithms of Mitchell [11, 12] carry over almost verbatim to the new setting of grid-rounded m-guillotine subdivisions. The main difference is in the complexity analysis.

A subproblem in the dynamic programming recursion is specified now by a rectangle ($O(n^4)$ choices), and, on each of the four sides, a segment corresponding to the 1-span ($O(n^2)$ choices per side), and a set of up to $2m$ M-grid points within each segment that specify the attachment points between this subproblem and neighboring subproblems. (Depending on the problem instance, other information, of constant size for fixed m, is also specified for a subproblem; see [12].) The key to the improvement given in this paper is that there are now only $\binom{M}{2m} = O(m^{4m})$ choices for these grid points on any one side, and this number is constant for fixed m. (Compare this to the $O(n^{2m})$ choices of crossing points in [12].) Thus, there are overall $O(n^{12})$ subproblems. We then optimize over all $O(n)$ choices of cuts, $O(n^2)$ choices of 1-spans along the cut, and $O(m^{4m})$ choices of grid points on the cut. The overall complexity of the dynamic programming algorithm is therefore $O(n^{15})$.

By rounding the 1-span intervals up to be intervals of lengths that are power-of-two factors smaller than the dimensions of the window, it is not hard to improve this complexity to $O(n^{10}\log^5 n)$, without significantly changing the approximation factor.

Corollary 1 *Given any fixed $\epsilon > 0$, and any set of n points in the plane, there is an $O(n^{O(1)})$ algorithm to compute a Steiner spanning tree (or Steiner k-MST), or a traveling salesperson tour, whose length is within a factor $(1+\epsilon)$ of minimum.*

S. Vempala (personal communication) has noted that if randomization is used in this method, the running time becomes (expected) $O(n\log^{O(1)} n)$. Details will appear in the full paper.

Acknowledgements

I thank Avrim Blum and Santosh Vempala for useful discussions.

References

[1] S. Arora, *Polynomial time approximation schemes for Euclidean TSP and other geometric problems*, Manuscript, March 30, 1996. Appears in Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci.(1996), pp. 2–12.

[2] S. Arora, *More efficient approximation schemes for Euclidean TSP and other geometric problems*, Unpublished manuscript, January 9, 1997.

[3] M. Bern and D. Eppstein, *Approximation algorithms for geometric problems*, In Dorit Hochbaum, editor, *Approximation Problems for NP-Complete Problems*, PWS Publications, 1997. Pages 296–345.

[4] A. Blum, P. Chalasani, and S. Vempala, *A constant-factor approximation for the k-MST problem in the plane*, Proc. 27th Annu. ACM Sympos. Theory Comput. (1995), pp. 294–302.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, Mass., 1990.

[6] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[7] D. Hochbaum, editor, *Approximation Problems for NP-Complete Problems*, PWS Publications, 1997.

[8] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[9] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *The Traveling Salesman Problem*, Wiley, New York, NY, 1985.

[10] C. Mata and J. S. B. Mitchell, *Approximation algorithms for geometric tour and network design problems*, Proc. 11th Annu. ACM Sympos. Comput. Geom. (1995), pp. 360–369.

[11] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-MST problem*, Proc. 7th ACM-SIAM Sympos. Discrete Algorithms (1996), pp. 402–408.

[12] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: Part II — A simple polynomial-time approximation scheme for geometric k-MST, TSP, and related problems*, SIAM J. Comp., to appear.
Available at `http://www.ams.sunysb.edu/~jsbm/`.

[13] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: Part III – Faster polynomial-time approximation schemes for geometric network optimization*, Manuscript, April 1997, available at `http://www.ams.sunysb.edu/~jsbm/`

[14] J. S. B. Mitchell, *Shortest Paths and Networks*, Chapter in the CRC Handbook of Computational Geometry, (E. Goodman and J. O'Rourke, eds.), CRC Press, to appear, 1997.

[15] J. S. B. Mitchell, A. Blum, P. Chalasani, and S. Vempala, *A constant-factor approximation for the geometric k-MST problem in the plane*, SIAM J. Comp., to appear. Available at `http://www.ams.sunysb.edu/~jsbm/`.

[16] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ, 1982.

Label Placement by Maximum Independent Set in Rectangles

Pankaj K. Agarwal* Marc van Kreveld† Subhash Suri‡

Abstract

Motivated by the problem of labeling maps, we investigate the problem of computing a large non-intersecting subset in a set of n rectangles in the plane. Our results are as follows. In $O(n \log n)$ time, we can find an $O(\log n)$-factor approximation of the maximum subset in a set of n arbitrary axis-parallel rectangles in the plane. If all rectangles have unit height, we can find a 2-approximation in $O(n \log n)$ time. Extending this result, we can find a $(1 + \frac{1}{k})$-approximation in time $O(n \log n + n^{2k-1})$ time, for any integer $k \geq 1$.

1 Introduction

Automated label placement is an important problem in geographic information systems (GIS), and has received considerable attention in recent years (for instance, see [4, 7]). The label placement problem includes positioning labels for area, line, and point features. The primary focus within the computational geometry community has been on labeling point features [3, 5, 14, 13]. A basic requirement in the label placement problem is that the labels be pairwise disjoint. Subject to this basic constraint, the most common optimization criteria are the number of features labeled and the size of the labels. Other variations include the choice of the shapes of the labels and the space of legal placements. Unfortunately, even in simple settings, the problem turns out to be NP-Complete or NP-Hard [2, 5].

In this paper we assume that each label is an orthogonal rectangle of fixed size and we want to place as many labels as possible. More precisely, let S be a set of n points in the plane. For each point $p_i \in S$, we have a label r_i, and a set π_i of marked points on the boundary of r_i; π_i may be a finite or an infinite set. Typical choices of π_i include the endpoints of the left edge of r_i, the four vertices of r_i, or the entire top and bottom edges of r_i. A valid placement of r_i is a translated copy $r_i + (p_i - x_i)$ of r_i for some $x_i \in \pi_i$, i.e., r_i is placed so that one of the marked positions on the boundary of r_i coincides with the point p_i. A *feasible configuration* is a family of pairs $\{(p_{i_1}, x_{i_1}), \ldots, (p_{i_k}, x_{i_k})\}$, where all the i_j are different and $x_{i_j} \in \pi_{i_j}$, so that the rectangles in $\{r_{i_1} + (p_{i_1} - x_{i_1}), \ldots, r_{i_k} + (p_{i_k} - x_{i_k})\}$ are pairwise disjoint. The *label placement* problem is to find a largest feasible configuration.

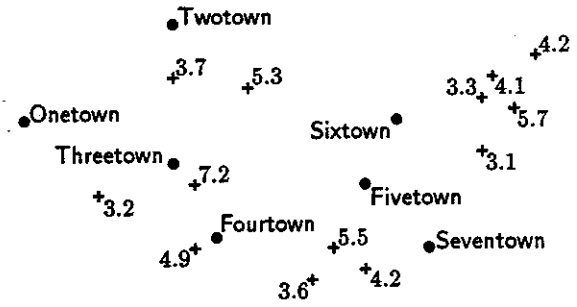


Figure 1: Point labels that are names of towns, mixed with epicenters of earthquakes labeled with their magnitude.

In practice the labels are subject to additional constraints, which help in simplifying and improving the algorithms. Restricting the shape of the labels to be same size squares is one such approach, as considered in [5, 14, 13], because in many technical maps all labels have the same size. Think of mapping measurements at sample points in a terrain, or maps showing magnitudes of earthquakes at points that are the epicenters. Another interesting case is when all labels have the same height but arbitrary width. This situation arises, for example, if we want to label city names on a map and all labels have the same font size, or when different types of point labels occur on a map. In this paper we consider the second case.

*Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129, USA. Research partially supported by National Science Foundation Grant CCR-93-01259, by an Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by an NYI award, and by matching funds from Xerox Corporation, and by a grant from the U.S.-Israeli Binational Science Foundation.

†Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands. Research partially supported by the ESPRIT IV LTR Project No. 21957 (CGAL).

‡Department of Computer Science, Washington University, St. Louis, MO 63130 USA. Research partially supported by NSF Grant CCR-9501494.

We will study the case when π_i has a constant number of positions on the boundary of r_i. Let $R_i = \{r_i + (p_i - x_j) \mid x_j \in \pi_i\}$ and set $R = \bigcup_{i=1}^{n} R_i$. The label placement problem is the same as computing a largest subset of pairwise disjoint rectangles in R. Since all rectangles in R_i have a common intersection point p_i, at most one rectangle can be chosen from each R_i. Consider the *intersection graph* $G(R)$ of R: the nodes of $G(R)$ are the rectangles of R and there is an edge between two nodes if the corresponding rectangles intersect. A subset of pairwise disjoint rectangles in R corresponds to an independent set in $G(R)$. We thus want to compute a maximum independent set of $G(R)$. Abusing the notation, we will say that we want to compute a maximum independent set of R. Computing an independent set of rectangles is known to be NP-Complete [6, 10]. This suggests that one should aim for approximation algorithms. We call an algorithm an *ε-approximation algorithm*, for $\varepsilon > 1$, if it returns an independent set of size at least γ/ε, where γ is the size of a maximum independent set of R.

Although it is known that no polynomial-time $\Omega(n^{1/4})$-approximation algorithm exists for maximum independent sets in arbitrary graphs [1], no such lower bound is known for intersection graphs of rectangles. In this paper we present an $O(n \log n)$-time $(\log n)$-approximation algorithm for rectangles.[1] If all rectangles in R have the same height, then we describe an $(1 + 1/k)$-approximation algorithm whose running time is $O(n \log n + n^{2k-1})$. This is an important case, since it models the label placement problem when all labels have the same font size. It is an open problem whether a c-approximation algorithm exists for arbitrary rectangles, for any positive constant c.

The paper is organized as follows. Section 2 summarizes the previous work on the label placement problem. In Section 3 we describe the approximation algorithm for arbitrary orthogonal rectangles. Section 4 describes our approximation algorithm for unit-height rectangles. Our algorithm is based on dynamic programming.

2 Previous Research

There has been a lot of work on label placement in cartography community; see e.g., [4, 7] and the references they contain for a sample of results. Algorithms researchers have also worked om labeling maps. Formann and Wagner [5] have studied the label placement for point features in the plane using *square* labels. Specifically, an axis-aligned square label is placed for each point such that the point coincides with one of the vertices of its labelling square. They used the *size* of the square label as the optimization criterion, subject to the condition that all points must receive a label. The square represents the text or measurement to be placed at the point. Their optimization is motivated by the maximum font size: since the problem allows scaling in the x-direction, it is the same as rectangular label placement for equal-size labels.

Given a point p, there are four positions for placing a square label so that the point coincides with one of the corners of the label. If all four positions of labels are allowed, then the problem of maximing the size of the label is NP-complete. Formann and Wagner give an $O(n \log n)$ time algorithm that guarantees a label size at least half the optimum [5]. They also show that no better approximation is possible unless P=NP. Formann and Wagner's approach is to grow all four possible labels around the points, removing candidate placements when they conflict with other growing labels. Whether the remaining labels allow a placement is done by solving 2-SAT problems. Kučera et al. [11] studied the same problem, but developed an exact, super-polynomial algorithm that can be applied for sets with up to roughly 100 points.

Wagner and Wolff [14, 13] have noted that, in practice, the approach of Formann and Wagner hardly ever results in square sizes significantly greater than half the optimum. They also study several variations and their implementation and find ways to improve on the size of the squares in practice.

Doddi et al. [3] allow more general shapes of labels, e.g., circles, nonoriented rectangles, ellipses, and present approximation algorithms in each case. Like Formann and Wagner, they are also approximating the size of labels. See also [9, 12].

Christensen et al. [2] provide a comparison of several approaches to place as many labels as possible on a map. They consider point labels, line labels, and area labels.

3 Arbitrary Size Rectangular Labels

We describe a simple, divide-and-conquer algorithm for computing a large independent set in a set R of n orthogonal rectangles in the plane. We sort the horizontal edges of R by their y-coordinates and their

[1] All logarithms in this paper are base 2.

vertical edges by their x-coordinates; this step takes $O(n \log n)$ time. This sorting is done only once in the beginning. If $n \leq 2$, we compute the maximum independent set in $O(1)$ time. Otherwise, we do the following.

1. Let x_{med} be the median x-coordinate among the abscissas of R.
2. Partition the rectangles of R into three groups: R_1, R_2, and R_{12}, where R_{12} contains rectangles intersected by the line $\ell : x = x_{med}$, and R_1 and R_2, respectively, contains the rectangles lying to the left and right of the line.
3. Compute I_{12}, the (real) maximum independent set of R_{12}. Recursively compute I_1, I_2, the approximate maximum independent sets in R_1 and R_2, respectively.
4. If $|I_{12}| \geq |I_1| + |I_2|$, return I_{12}, otherwise return $I_1 \cup I_2$.

The key insight behind the algorithm is that since all rectangles in R_{12} intersect the line ℓ, it suffices to compute a largest nonoverlapping subset of intervals in the set $J = \{r \cap \ell \mid r \in R_{12}\}$, in order to compute I_{12}. This one-dimensional rectangle independent-set problem can be solved optimally by the following greedy strategy in $O(n \log n)$ time. Sort the intervals in the ascending order of their right endpoints. Add the leftmost interval ℓ to the independent set; delete all intervals intersecting ℓ; and repeat until no more intervals are left. Recall that the vertical edges of rectangles in R are sorted by their x-coordinates, so we can sort the intervals in J by their right endpoints in linear time. Since $|R_1|, |R_2| \leq |n|/2$, the overall running time of the algorithm is $O(n \log n)$.

Next, we prove that our algorithm computes an independent set of size at least $\gamma / \max(1, \log n)$, where γ is the largest independent set. For $n \leq 2$, we compute a largest independent set, so the claim is obviously true for $n \leq 2$. Suppose it is true for all $m < n$. Let I^* be a maximum independent set of R. Set $I_1^* = I^* \cap R_1$, $I_2^* = I^* \cap R_2$, and $I_{12}^* = I^* \cap R_{12}$. Since the algorithm computes a maximum independent set I_{12} of R_{12}, we have $|I_{12}| \geq |I_{12}^*|$. By induction hypothesis,

$$|I_1| \geq \frac{|I_1^*|}{\log(n/2)} \geq \frac{|I_1^*|}{\log n - 1} \quad \text{and} \quad |I_2| \leq \frac{|I_2^*|}{(\log n - 1)}.$$

Therefore

$$|I| = \max\{|I_{12}|, |I_1| + |I_2|\} \geq$$

$$\max\left\{|I_{12}^*|, \frac{|I_1^*| + |I_2^*|}{\log n - 1}\right\} \geq$$

$$\max\left\{|I_{12}^*|, \frac{|I^*| - |I_{12}^*|}{\log n - 1}\right\} \geq \frac{|I^*|}{\log n},$$

as desired. Hence, we obtain the following result.

Theorem 1 *Let R be a set of n axis-parallel rectangles in the plane. An independent set of of R of size at least $\gamma / \log n$ can be computed in time $O(n \log n)$, where γ is the size of a maximum independent set in R.*

4 Approximation Scheme for Unit-Height Rectangles

In this section we develop a polynomial-time approximation algorithm for computing an independent set of rectangles of fixed height, but of arbitrary width. As discussed earlier, our class is clearly more general that unit squares, and it is of particular interest to labeling maps. We assume without loss on generality that all rectangles have unit height. We first develop a 2-approximation algorithm, which takes $O(n \log n)$ time. Then, using dynamic programming, we obtain an γ-approximation algorithm whose running time is $O(n \log n + n^{2\gamma - 1})$ time for $\gamma \geq 1$.

4.1 A 2-approximation algorithm

Consider a set R of n unit-height rectangles in the plane. We draw a set of horizontal lines, $\ell_1, \ell_2, \ldots, \ell_m$, where $m \leq n$, so that the following three conditions hold.

1. The separation between two lines is strictly more than one,
2. each line intersects at least one rectangle, and
3. each rectangle is intersected by some line.

Note that minimum separation condition implies that a rectangle cannot be intersected by more than one line. The lines can be drawn from top to bottom using an incremental approach. These lines partition the set R into subsets $R_1, R_2, \ldots, R_m$, where R_i is the set of rectangles in R that intersect line ℓ_i.

We compute a maximum independent set M_i for each R_i, which takes $O(|R_i| \log |R_i|)$ time, using the one-dimensional greedy algorithm. Since the line ℓ_i does not intersect any rectangle of $R \setminus R_i$, the rectangles in M_i do not intersect any rectangle of M_j except for $j = i - 1$ or $j = i + 1$. Consider the two independent sets $\{M_1 \cup M_3 \cup \cdots \cup M_{2\lceil m/2 \rceil - 1}\}$ and $\{M_2 \cup M_4 \cup \cdots \cup M_{2\lfloor m/2 \rfloor}\}$. Clearly, the larger of these two must have size at least $\gamma/2$, and thus

we have a 2-approximation algorithm. The running time of the algorithm is $O(n \log n)$, since finding the lines ℓ_i and forming the corresponding partition can be done in a single pass through the rectangles after sorting them by their y-coordinates.

Theorem 2 *Let R be a set of n unit height axis-parallel rectangles in the plane. In $O(n \log n)$ time, we can compute an independent set of size at least $\gamma/2$, where γ is the size of a maximum independent set of R.*

4.2 A $(1 + \frac{1}{k})$-approximation algorithm

We now combine dynamic programming with the shifting technique of Hochbaum and Maass [8] to improve the approximation factor to $(1 + \frac{1}{k})$, for any $(1 + \frac{1}{k}) \geq 1$. The basic idea is to partition the rectangles by horizontal lines $\ell_1, \ell_2, \ldots, \ell_m$ as before, but then use dynamic programming to *optimally* solve the subproblem for each set of rectangles intersected by γ consecutive lines. Suppose the lines are labeled $\ell_1, \ell_2, \ldots, \ell_m$ from top to bottom, and R_i is the set of rectangles intersected by line ℓ_i. Define $R_i^k = R_i \cup R_{i+1} \cdots R_{i+k-1}$, that is, R_i^k is the set of rectangles intersected by any line in the set $\{\ell_i, \ell_{i+1}, \ldots, \ell_{i+k-1}\}$. We will refer to R_i^k's as *subgroups*. We now define $k+1$ groups $G_1, \ldots, G_{k+1}$, where

$$G_j = \bigcup_{i \geq 0} R^k_{i(k+1)+j} \quad = \quad R \setminus \bigcup_{i \geq 0} R_{i(k+1)+j}.$$

That is, for $1 \leq j \leq k+1$ the group G_j is obtained from R by deleting rectangles intersected by every $(k+1)$-st line, starting with ℓ_j. We make two key observations about these groups of rectangles. First, consider two consecutive subgroups within any group, such as R_1^k and R_{k+2}^k in G_1. No rectangle of R_1^k intersects a rectangle in R_{k+2}^k; the line ℓ_{k+1} separates these subgroups. By extension, this means that rectangles in a subgroup are disjoint from the rectangles of any other subgroup in the same group. Thus, if we combine the independent sets for all the subgroups, we get an independent set of that group. Second, since a group is formed by deleting all rectangles that intersect every $(k+1)$-st line, all rectangles in $R \setminus G_i$ are intersected by at most $\lfloor m/(k+1) \rfloor$ lines. Thus, if we compute a maximum independent set for each G_i, and choose the largest one, we can miss at most $\gamma/(k+1)$ rectangles. Hence we get an $(1 + \frac{1}{k})$ factor approximation. This is exactly the shifting idea of Hochbaum and Maass [8], and this is precisely what we will do as well.

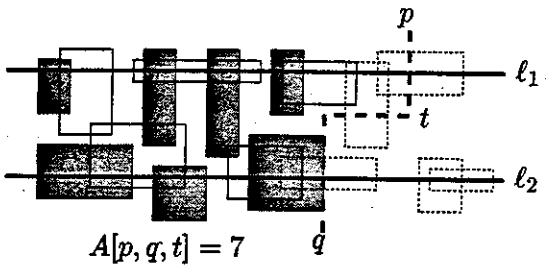


Figure 2: Polygonal line defined by p, q, t and its relation to the table entry $A[p, q, t]$.

We give a dynamic programming solution for computing a maximum independent set $M(R_j^k)$ for any subgroup R_j^k, that is, a set of rectangles intersected by k consecutive lines in $\ell_1, \ell_2, \ldots, \ell_m$. After computing $M(R_j^k)$ for every $j \geq 1$, the rest of the algorithm is rather straightforward.

For ease of exposition, we describe the algorithm for the case $k = 2$, but all the ideas generalize readily. Without loss of generality, let us consider the problem of computing a maximum independent set for $R_1^2 = R_1 \cup R_2$, that is, the rectangles intersected by ℓ_1 or ℓ_2. Let $X = (x_1, x_2, \ldots, x_m)$ denote the sequence of distinct abscissas, sorted in the increasing order (left to right), and let $Y = (y_1, y_2, \ldots, y_h)$ denote the sequence of distinct ordinates in $R_1 \cup R_2$, sorted in the decreasing order (top to bottom).

With each triple $\tau = (p, q, t)$, where $p, q \leq m$ and $t \leq h$, we associate a polygonal line λ_τ, defined as follows: If $p = q$, then λ_τ is the vertical line $x = p$; otherwise λ_τ consists of a vertical ray emanating from the point (x_p, y_t) in the $(+y)$-direction, the horizontal segment connecting (x_p, y_t) to (x_q, y_t), and another vertical ray emanating from the point (x_q, y_t) in the $(-y)$-direction; see Figure 2. Let $R_\tau \subseteq R$ denote the set of rectangles whose interiors lie to the left of the line λ_τ. Let M_τ denote a maximum independent set of R_τ, and let $A_\tau = |M_\tau|$. We now describe how we compute A_τ for all triples $\tau = (p, q, t)$. We will construct a three-dimensional table A, in which $A[p, q, t]$ will store the value of $A_{(p,q,t)}$.

We will consider the case when $p > q$; the case $p < q$ is symmetric. If $p = q$, the third index t plays no role. In this case, we try two choices for t: the y coordinate of ℓ_1 and the y coordinate of ℓ_2. The algorithm for $p > q$ can be modified to handle this case as well.

If $p > q$ and no rectangle in $R_\tau \cap R_1$ has its right edge at $x = x_p$, then $R_\tau = R_{(p-1,q,t)}$; therefore $A[p, q, t] = A[p-1, q, t]$. Otherwise, let $r \in R_1$ be the rectangle whose right edge is at $x = x_p$. (Let us assume that there is only such rectangle; we will

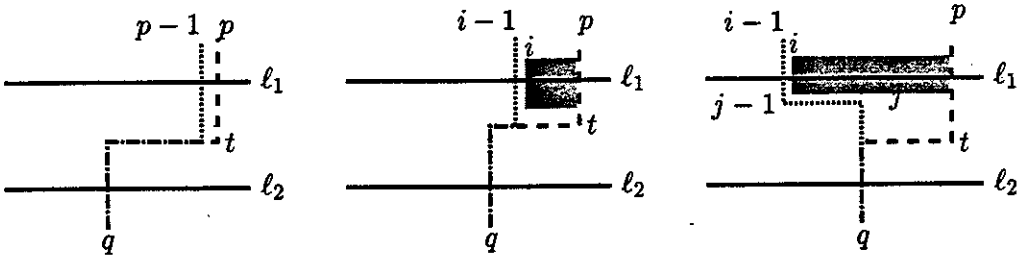


Figure 3: Filling in the entry $A[p, q, t]$; the three cases.

$$A[p,q,t] = \begin{cases} A[p-1,q,t] & \text{no rectangle in } R_{(p,q,t)} \cap R_1 \text{ has the right edge at } x = x_p; \\ \max(A[p-1,q,t], & R_{(p,q,t)} \cap R_1 \text{ has a rectangle } r \text{ with the right edge at } x = x_p, \\ \quad A[i-1,q,t]+1) & \text{the left edge at } x = x_i, \text{ and } i > q; \\ \max(A[p-1,q,t], & R_{(p,q,t)} \cap R_1 \text{ has a rectangle } r \text{ with the right edge at } x = x_p, \\ \quad A[i-1,q,j-1]+1) & \text{the left edge at } x = x_i \text{ with } i \le q, \text{ and bottom edge at } y = y_j. \end{cases}$$

Figure 4: Recursive definition of the entry $A[p, q, t]$.

discuss later how to handle the case when the right edges of many rectangles lies on the line $x = x_p$.) Suppose the left edge of r lies on the line $x = x_i$ and its bottom edge lies on the line $y = y_j$. If $r \notin M_\tau$, then again $A[p, q, t] = A[p-1, q, t]$. On the other hand, if $r \in M_\tau$, then none of the other rectangles in R_τ that intersect r can belong to M_τ. If $x_i > x_p$, then let $\tau' = (i-1, q, t)$, otherwise let $\tau' = (i-1, q, j-1)$. It is easy to see that if $t \in M_\tau$, then $M_\tau = M_{\tau'} \cup \{r\}$. Therefore, $A_\tau = A_{\tau'} + 1$. Hence, we obtain the following recurrence for $A[p, q, t]$, assuming that $p > q$.

The entries $A[p, q, t]$ are recursively computed as in Figure 4.

If there are many rectangles in $R_\tau \cap R_1$ touching the line $x = x_p$, we divide them into two subsets—the ones whose left edge lies to the left of $x = x_q$ and the ones whose left edge does not lie to the left of $x = x_q$. For each rectangle in the first category, we use the third case and for all rectangles in the second category we apply the second case. We then choose the one that gives the maximum value. We can fill out the three-dimensional table A in a standard dynamic programming manner. Geometrically, the only constraint on filling out the entries is that when $A[p, q, t]$ is being computed, we must have computed the entries corresponding to the polygonal lines that lie in the closure of the subplane left of $\lambda_{(p,q,t)}$. A straightforward implementation of the dynamic program requires $O(|R_1 \cup R_2|^3)$ time—most entries take constant time, except when several rectangles have their right edge at the same p or q. However, we can afford to spend time proportional to the number of rectangles, since the total work still adds up to $O(|R_1 \cup R_2|^3)$.

Let $|R_i| = n_i$, for $i = 1, 2, \ldots, m$, where recall that m is the number of lines used to partition R. Then, clearly $\sum_{i=1}^{m} |R_i| = \sum n_i = n$. In order to compute an independent set of size $2\gamma/3$, we perform the dynamic programming algorithm $m-1$ times, once for each pair of consecutive lines. Thus the total time complexity is

$$\sum_{i=1}^{m-1} O((n_i + n_{i+1})^3) = O(n^3).$$

Observe that that if $n_i = O(\sqrt{n})$ for all i—a situation that is likely in practice—then the running time is only $O(n^2)$. It is straightforward to adapt the algorithm so that it computes the independent set rather than the size of it.

Theorem 3 *Let R be a set of n unit-height axis-parallel rectangles in the plane. In $O(n^3)$ time, we can compute an independent set of size at least $2\gamma/3$, where γ is the size of a maximum independent set of R.*

Extending the technique to a $(1 + \frac{1}{k})$-approximation algorithm is straightforward. We need to compute the an optimum solutions for the union of rectangles intersecting k consecutive lines. In the dynamic programming algorithm,

instead of a 3-dimensional table, we need to fill out a $(2k-1)$-dimensional table. Geometrically, a $(2k-1)$-tuple corresponds a polygonal line, which is a weakly y-monotone, rectilinear polyline with two vertical half-lines, $k-2$ horizontal edges, and $k-3$ vertical edges. Each vertical edge has its x-coordinate in X, and each horizontal edge has its y-coordinate in Y. This gives us the polynomial-time approximation scheme with the following performance.

Theorem 4 *Let R be a set of n unit-height axis-parallel rectangles in the plane. In $O(n^{2k-1})$ time, we can compute an independent set of size at least $\gamma/(1+\frac{1}{k})$, for any $k \geq 1$, where γ is the size of a maximum independent set of R.*

5 Conclusions

We have given approximation algorithms and an approximation scheme for maximum size non-intersecting subset in sets of rectangles. The work is motivated from label placement at points, where the rectangles represent the bounding boxes of labels. The approximation scheme was known for the restrictive case of unit size square labels, which occurs for fixed precision decimal numbers as labels. We gave a different approximation scheme for unit height labels with varyings widths, which is the standard situation for labels that are names, or labels of different type with fixed font size.

The algorithms for labeling supported the situation where several positions for the label of any point are allowed. The restriction is that all positions of the label of a point intersect each other. Also, the running time is not affected if a constant number of positions is allowed for each label.

The maximum non-intersecting subset of rectangles problem can be seen as a maximum independent set problem in a special type of graph. The approximation algorithm we presented for these graphs is considerably better than what is theoretically possible for general graphs. However, we were not able to obtain a polynomial time, constant factor approximation algorithm for the case of arbitrary axis-parallel rectangles. This is an interesting open problem.

References

[1] M. Bellare and M. Sudan. Improved non-approximability results. In *Proc. 26th Symp. Theory of Computing*, pages 184–193, 1994.

[2] J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Trans. Graphics*, 14:202–232, 1995.

[3] S. Doddi, M.V. Marathe, A. Mirzaian, B.M.E. Moret, and B. Zhu. Map labeling and its generalizations. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, 1997.

[4] J.S. Doerschler and H. Freeman. A rule-based system for dense-map name placement. *Comm. ACM*, pages 68–79, 1992.

[5] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 281–288, 1991.

[6] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, 12(3):133–137, 1981.

[7] H. Freeman. Computer name placement. In D. J. Maguire, M. F. Goodchild, and D. W. Rhind, editors, *Geographical Information Systems: Principles and Applications*, pages 445–456. Longman, London, 1991.

[8] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32:130–136, 1985.

[9] H.B. Hunt III, M.V. Marathe, V. Radhakrishnan, S.S. Ravi, D.J. Rosenkrantz, and R.E. Stearns. A unified approach to approximation schemes for NP- and PSPACE-hard problems for geometric graphs. In *Proc. 2nd Europ. Symp. on Algorithms*, volume 855 of *Lect. Notes in Comp. Science*, pages 424–435, 1995.

[10] H. Imai and Ta. Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J. Algorithms*, 4:310–323, 1983.

[11] L. Kučera, K. Mehlhorn, B. Preis, and E. Schwarzenecker. Exact algorithms for a geometric packing problem. In *Proc. 10th Sympos. Theoret. Aspects Comput. Sci.*, volume 665 of *Lecture Notes in Computer Science*, pages 317–322. Springer-Verlag, 1993.

[12] M.V. Marathe, H. Breu, H.B. Hunt III, S.S. Ravi, and D.J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995.

[13] F. Wagner and A. Wolff. An efficient and effective approximation algorithm for the map labeling problem. In *Proc. 3rd Europ. Symp. on Algorithms*, volume 979 of *Lect. Notes in Comp. Science*, pages 420–433, 1995.

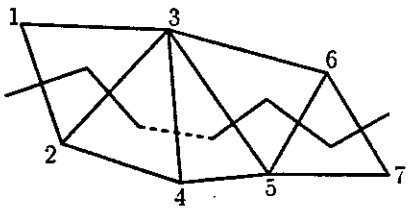
[14] Frank Wagner and Alexander Wolff. Map labeling heuristics: Provably good and practically useful. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 109–118, 1995.

Easy triangle strips for TIN terrain models*

Bettina Speckmann Jack Snoeyink
Dept. of Computer Science
University of British Columbia

1 Introduction

The triangulated irregular network (TIN) [5] is one of the basic models for representing digital terrains. The common bottleneck for GIS applications that display TINs is the rate at which triangulation data can be sent into the graphics engine. *Tristrips* can be used to reduce the amount of data that must be sent.



without swaps: $(1, 2, 3, 4)(4, 3, 5, 6, 7)$
with swaps: $(1, 2, 3, 4, 3, 5, 6, 7)$

Figure 1: Triangle strips

Each triangle in a TIN uses three data points. If the triangles are ordered so that consecutive triangles share an edge, then it is necessary to specify only the incremental change of one vertex per triangle. The resulting list of vertices constitutes a triangle strip or *tristrip* (Fig. 1), which is supported by the OpenGL graphics library. The use of tristrips can potentially reduce amount of data to be transmitted, and hence the rendering time, by a factor of three. A triangle strip requires, however, that all turns alternate from left to right. To obtain two consecutive left or right turns a vertex must be "swapped", i.e. transmitted twice, creating an empty triangle (Fig. 1).

Given a triangulation having m triangles, the theoretical lower bound on the number of vertices in a tristrip is $m + 2$; this bound is attained only by "sequential triangulations" [1]—triangulations whose dual graphs contain a Hamiltonian path in which no three consecutive triangulation edges crossed by the path are incident upon the same triangulation vertex. For other triangulations, including those common in representing digital terrains, a tight lower bound has not been proved. Heuristics are required to find "good" tristrips that use a small number of vertices to represent a given terrain.

Arkin et al. proved that a depth-first traversal of any spanning tree of the dual graph of a triangulation results in a triangle strip representation that uses at most $9m/4$ vertices [1]. We observe experimentally that choosing a special spanning tree, namely the one induced by the traversal algorithm in [2], and traversing it in a modified depth-first fashion, we construct triangle strips that use less than $3m/2$ vertices to represent a TIN. Using this special spanning tree is an easy and fast way to construct tristrips that requires no modification of the TIN and no additional storage.

We report on the number of vertices and time taken to compute tristrips from different numbers of triangles. We also report on several approaches to decrease the number of vertices used to

*Supported in part by an NSERC Research Grant, B.C. Advanced Systems Institute, and Facet Decision Systems.

represent a given triangulation, including the insertion of empty triangles to facilitate longer tristrips (swapping) and the connection of single triangles from different tree branches.

Evans, Skiena and Varshney developed an algorithm for constructing tristrips from partially triangulated models [3]. Their algorithm can handle fully triangulated models, like a TIN, and produces 3–5% fewer vertices using a technique called patchification. However, on typical TIN sizes, their algorithm is 100–1000 times slower than our method on the same data sets and hardware.

In the following sections we first give a short summary of the traversal algorithm in [2] and then show how this algorithm can be used to efficiently construct tristrips. We conclude with experimental results.

2 Traversing the TIN

The traversal algorithm of [2], which extends work of [4], establishes a visibility order on the triangles and visits them in this order.

For each triangle of triangulation T, it defines an adjacent *predecessor* triangle and forms a directed graph $G(V, E)$ with $V = \{t \mid t \in T\}$ and $E = \{(t', t) \mid t' \textit{is the predecessor of } t\}$. Graph G is actually a directed tree, rooted at a distinguished *starting triangle* t_{start}. The basic traversal visits the triangles of the tree G in depth-first order. Although G provides the traversal order for the triangulation, G is never explicitly determined or stored.

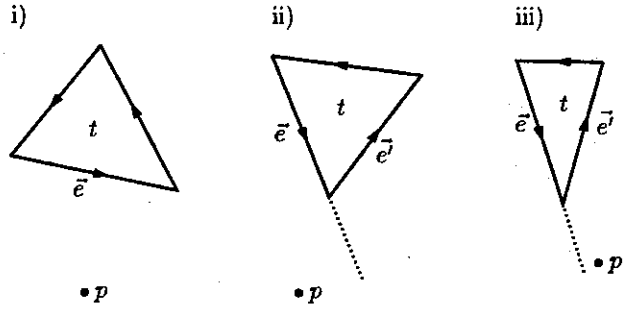


Figure 2: The predecessor of t is adjacent to edge e in cases i) and ii) and e' in case iii).

To define the *predecessor* relation, first choose an arbitrary point p in the starting triangle t_{start}. Then, for any triangle $t \in T - \{t_{start}\}$, compute the point of t that is closest to p under Euclidean distance. If the closest point is inside an edge e of t (Fig. 2.i), then the predecessor of t is the other triangle t' that is also adjacent to e. Otherwise, the closest point is a vertex of t; orient the edges of t counterclockwise, and consider the edges e and e' that are just before and just after this vertex. If e is exposed to p (i.e., the directed line induced by $\vec{e}$ has p strictly to the right), then the algorithm chooses the triangle adjacent to e as the predecessor of t (Fig. 2.ii), otherwise it chooses the triangle adjacent to e' (Fig. 2.iii).

The graph G induced by this predecessor relationship has $m - 1$ edges and is connected—there is a path from any triangle to t_{start}—therefore, it is a tree (Fig. 3).

3 Constructing tristrips

Given any spanning tree of the dual of a triangulation the basic method to construct tristrips follows the tree in a depth-first manner ans starts a new tristrip whenever the sequence of left and

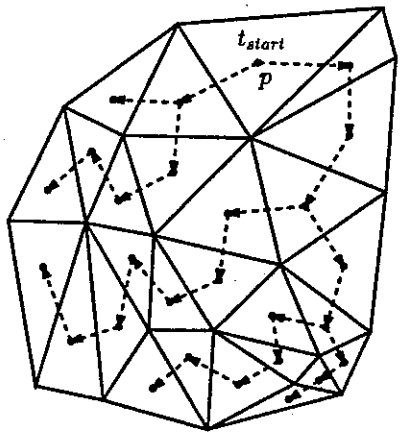
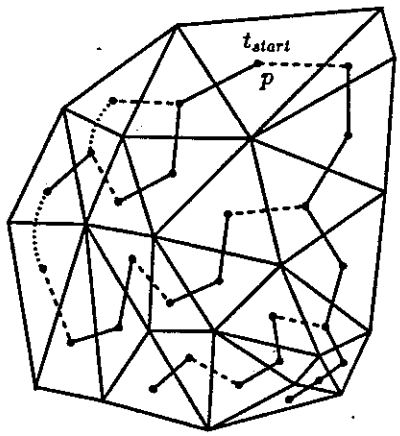


Figure 3: The spanning tree induced by the predecessor relationship



triangle strips
single triangles can be added to a strip

Figure 4: Partitioning the spanning tree into tristrips

right turns is violated. For triangulations with m triangles, Arkin et al. proved that depth-first traversal of any spanning tree results in a tristrip representation that uses $9m/4$ vertices [1].

We observed, however, that branches of the spanning tree defined in the previous section typically turn from left to right as one follows a path down from the root. Furthermore, no information about this spanning tree has to be stored explicitly; the algorithm decides with which child of a triangle t to proceed based on geometric calculations on the neighbors of t. Since the predecessor of a triangle can be computed in constant time, this algorithm computes the tristrip representation using linear time in the number of triangles.

Having constructed the basic tristrips, it is easy to insert "left-over" single triangles into already existing strips, even if they are not connected via the spanning tree, by just traversing the list of tristrips (Fig. 4). If one wanted to minimize the amount of time needed, however, it would be necessary to maintain an extra data structure recording the tristrip that each triangle of the TIN is assigned to. The current implementation does not add extra data structures to the TIN, therefore the time used to insert single triangles is not linear in the number of triangles (Tab. 2).

There are several other possibilities to decrease the number of vertices in a basic tristrip representation. The first is to allow *swaps*, i.e. to allow two consecutive left or right turns. For each swap a vertex has to be transmitted a second time, but, in return, two tristrips can be joined, saving two vertices. So a swap decreases the number of vertices used in the tristrip representation by one (Fig. 5). Swaps can be chosen during the traversal without additional data structure; "Left-over" single triangles can also be added as mentioned above.

A second way looks for nodes where the tree is "very wide:" i.e. both the right and the left child of a node exist, the right child has its own right child, and the left child has its own left child. One can create a tristrip that combines the two strips starting at the right and left child of the node, saving additional vertices depending on the shape of the spanning tree (Fig. 6). This "strip combining" modifies the traversal locally but still does not require additional data structures and

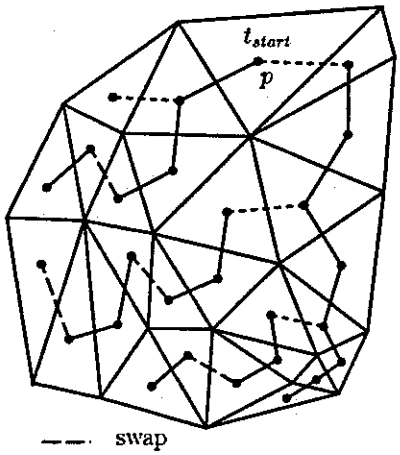


Figure 5: Allowing swaps in the construction of tristrips

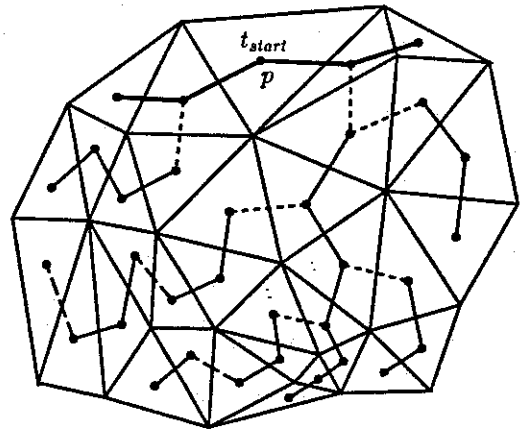


Figure 6: Combining the tristrips starting at the left and right child of t_{start}

can be used with either the basic traversal or the traversal using swaps. Again, single triangles can be added to the constructed strips.

A third way would be to join strips using non-tree edges, either with or without swaps. Our attempts at doing so were hampered by the size of the TINs that we wanted to be able to handle and by the desire to handle them without adding data structure to the TIN itself.

4 Experimental Results

We have implemented this algorithm in Facet, a GIS product of Facet Decision Systems, and tested it and its variations on a number of large TIN terrain models.

Table 2 lists the number of vertices for the variations that were discussed. Fig. 7 illustrates the number of vertices per triangle for the basic traversal and each variation. Introducing swaps has the most significant impact, reducing the number of vertices by 5–7% over the basic traversal algorithm. Tree combining does not help significantly. Joining singles to adjacent tristrips helps the basic method, but takes more time—because we do not leave markers behind in the TIN structure, our implementation searches ends of tristrips to attach singles. It helps the "swaps" method by only 0.5%, so we will use the swaps method alone in the next comparison.

number of triangles	number of vertices	number of tristrips	time in sec.
814	1182	184	0.02
2505	3628	563	0.04
27578	39512	5967	1.2
52610	74520	10955	2.3
86674	123420	18373	2.8
101290	146522	22616	3.6
170032	245756	37862	6

Table 1: Number of vertices, tristrips, and seconds for seven triangulations

Table 3 compares the "swap" method with "stripe" [3]. We see that, on larger TINs, stripe produces 3–5% fewer vertices to represent the same triangulation. It does relatively better on smaller TINs. Unfortunately, stripe takes a considerable amount of time. The graph in Fig. 8

number of triangles	number of vertices used to represent triangulation: basic w/ singles	combine	combine w/ singles	swap	swap w/ singles	time in sec.	with singles
814	1148	1178	1148	1132	1124	0.02	0.03
2505	3540	3622	3532	3454	3428	0.04	0.3
27578	38970	39486	38942	37543	37392	1.2	4
52610	73604	74492	73580	70968	70704	2.3	22
86674	121672	123332	121576	117685	117123	2.8	26
101290	144588	146391	144454	139261	138725	3.6	35
170032	242568	245502	242292	233746	232768	6	85

Table 2: Number of vertices using variations of the basic method on several triangulations

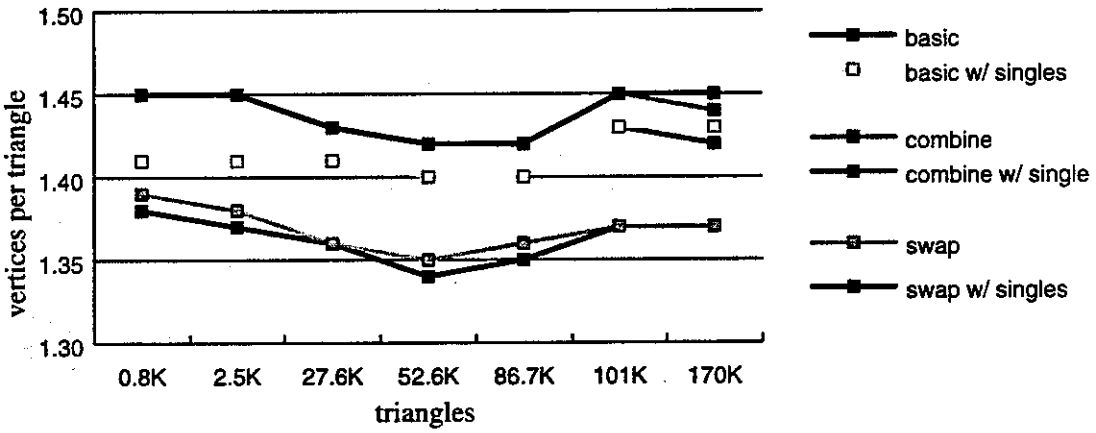


Figure 7: Number of vertices per triangle for our variations

shows the differences in number of vertices and miliseconds spent per triangle. Our traversal is not only fast, but the time scales linearly with the problem size. (From the table, we note that stripe produces fewer tristrips, apparently using extra swaps to join fragments that our method does not.)

These experiments were performed on a 170Mhz SUN UltraSPARC, with 128Mb of memory using digital elevation data from the British Columbia TRIM standards for the Weaver Lake, BC area, and the Vancouver BC north shore. The stripe program ran out of memory and could not complete the computation for the TIN with 187872 triangles.

Acknowledgments

We thank Facet Decision Systems for the use of their system and access to data. We especially thank Gerry Furseth for discussions and for adding tristrips into the Facet Visualizer.

References

[1] E. M. Arkin, M. Held, J. S. B. Mitchell, and S. S. Skiena. Hamiltonian triangulations for fast rendering. In J. van Leeuwen, editor, *Algorithms – ESA '94*, volume 855 of *Lecture Notes Comput. Sci.*, pages 36–47, Sept. 1994.

number of triangles	swaps # of vertices	swaps # of tristrips	swaps time in sec.	stripe # of vertices	stripe # of tristrips	stripe time in sec.
814	1132	112	0.02	963	29	0.52
1502	2064	194	0.04	1848	46	1.32
2502	3454	337	0.07	2899	81	2.54
5686	7835	724	0.23	7480	165	11.83
10810	14744	1352	0.44	14037	334	40.33
27578	37543	3430	1.07	36046	821	311.90
52610	70968	6280	1.88	68562	1455	1280.13
97930	134366	12058	3.27	128096	2848	4734.07
187872	280580	31223	7.19			

Table 3: Comparing our "swaps" algorithm with "stripe" [3] on several triangulations

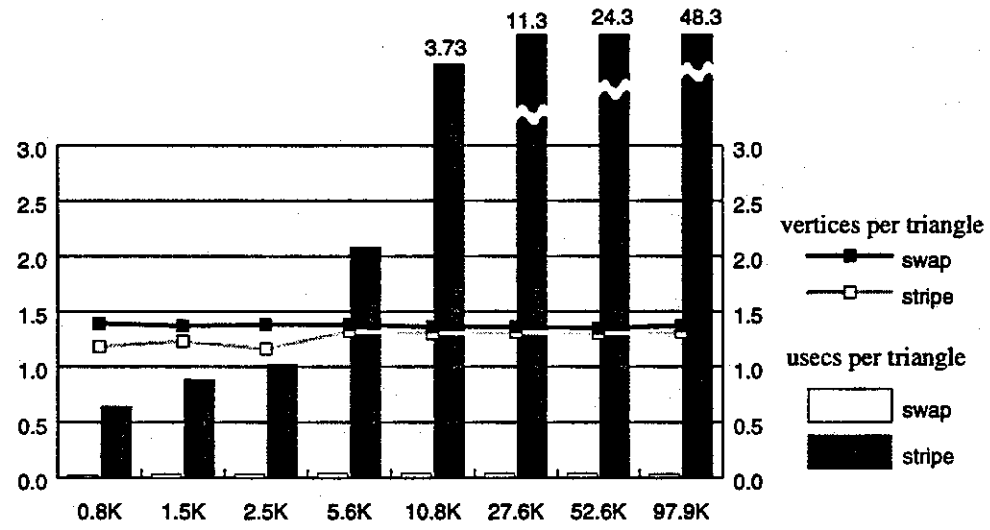


Figure 8: Number of vertices and running time (μsecs) per triangle

[2] M. de Berg, M. van Kreveld, R. van Oostrum, and M. Overmars. Simple traversal of a subdivision without extra storage. *Int. J. of GIS*, 11, 1997.

[3] F. Evans, S. S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *IEEE Visualization '96*. IEEE, Oct. 1996. ISBN 0-89791-864-9.

[4] C. M. Gold, T. D. Charters, and J. Ramsden. Automated contour mapping using triangular element data structures and an interpolant over each irregular triangular domain. *Computer Graphics*, 11(2):170–175, July 1977. Proc. SIGGRAPH '77.

[5] T. K. Peucker, R. J. Fowler, J. J. Little, and D. M. Mark. The triangulated irregular network. In *Amer. Soc. Photogrammetry Proc. Digital Terrain Models Symposium*, pages 516–532, 1978.

Partitioning Algorithms for Transportation Graphs and Their Applications to Routing *

Cavit Aydin Doug Ierardi

Department of Computer Science
University of Southern California

Abstract

Real transportation graphs have distinct characteristics that make specialized and efficient partitioning techniques. Such partitions have proved to be essential in minimum cost path algorithms for intelligent transportation systems (ITS), for example, where queries are under time constraints and the underlying graphs are typically very large. Previous studies, however, do not explicitly make use of the distinct characteristics — both graph theoretic and statistical — that are found in real transportation graphs. In this abstract, we focus on a class of minimum–cost path queries on transportation graphs that rely on partitioning to optimize their performance. We approach the problem in two ways: first, we consider algorithms with provably good complexity bounds; next, we apply these algorithms to real transportation graphs (based upon the entire database underlying the Thomas Brothers guide of the Los Angeles area), with further practical simplifications suggested by statistical properties of these graphs.

1 Introduction

Natural problems that arise in the design of intelligent transportation systems (ITS) and geographical information systems (GIS) have gained attention from a variety of academic disciplines. These new domains have generated several important problems to be studied and solved. One such problem, which has been identified as a key requirement for ITS, is the efficient processing of path queries. The motivation is that an effective routing system can minimize the travel time, reduce the energy consumption, regulate the traffic flow and potentially decrease the number of traffic accidents.

This research partially supported by the National Science Foundation under grants CCR–9402819 and ECS–9510656 and ISLA/IDA project at USC.

Different underlying architectures have been proposed for such systems. In a *decentralized* system, each vehicle has its own copy of the transportation map, typically stored on a CD-ROM, and a processor for computing paths. Such systems have the disadvantage of not being able to implement global optimizations with respect to changing traffic conditions. On the other hand, in a *centralized* system, only a dedicated computing center keeps the transportation maps and handles all the queries. The system itself, however, might be parallelized or distributed. (See [15] for an example.)

The route guidance conducted by a centralized system can further be classified into *path-based* and *direction-based*. In the path-based case, after receiving a query from a vehicle, the system computes an optimized path and transmits the entire path to the vehicle. The vehicle can then follow the path with on-board navigation device without further queries to the system. However, such a system has the disadvantage that the originally computed path may no longer be optimal under changing traffic conditions. In a direction-based guidance system, only the path to the next intersection (or "beacon" [15]) along the current shortest path to the destination is transmitted to the vehicle. Hence the system does not need to compute the entire path for each query. Such an approach can quickly adjust the routes under changing road conditions. A centralized direction-based ITS design is presented in [15].

For direction-based systems, the essential problem is to find the next optimal direction to be taken from the source to destination on a given transportation map (with costs assigned through some metric, such as distance or expected time). More precisely, we'll let $G = (V, E)$ be an n-vertex graph. Given a source node $v_1 \in V$ and destination node $v_2 \in V$, we would like to find its successor $v_3 \in V$ such that $(v_1, v_3) \in E$ is on the minimum weight

path from v_1 to v_2.

The necessity of efficient shortest path algorithms for transportation applications motivates special–case improvements to some well known graph theoretic algorithms. The nature of the solution should also change when we consider the nature of the proposed architectures and the actual sizes of such graphs and the resources expected for such systems. For example, the algorithm which we advocate below utilizes a rather large-scale precomputation, requiring a large amount of additional in-memory storage. However, even for Los Angeles county — a map with about 180,000 nodes — it provides an attractive and feasible solution for a centralized system.

1.1 Related Work

There have been extensive study on the shortest path problem, both in theory and in practice. For all-pairs shortest path problems, the classic algorithms are Floyd-Warshall for general graphs and Johnson's algorithm for sparse graphs [4]. For the single-source shortest path problem the there are Bellman-Ford and Dijkstra's algorithm [4]. Parallel and distributed algorithms have also been developed. For example, in [7], a graph is fragmented to recursively decompose the problem into smaller tasks; these tasks are in turn assigned to different processors. However these methods are designed to work with acyclic subgraphs of initial graph. This restriction makes it less suitable for transportation graphs which are by nature highly cyclic.

Recent work attempts to find efficient solutions to the single pair shortest path problem, one of the most important for ITS. However, in many respects the character of these problems differs from the classical treatment: The graph itself, together with cost information, is now more like a relatively static database, on which one must process queries. And, although the underlying geometric and geographic databases may be very large, topological information for path queries is often relatively compact. For example, the transportation graph of all Los Angeles County has only 177,748 nodes and 494,452 edges. Together, these requirements make memory-resident databases attractive for centralized systems.

Shekar et al. developed a hierarchical A* algorithm in [14]. Their method is a heuristic which takes advantage of the existence of high-speed roads like freeways and highways. In this model each node in the graph has a fixed entry/exit node onto the high-speed links based on the shortest geographic distance. After a vehicle reaches to the high-speed links they stay on them until the destination is reached.

Agrawal et al. [1] and Huang et al. [8] develop a path encoding method where each node stores the tuple $\langle destinaton, successor, weight \rangle$ for all the destination nodes in the graph. This gives an $O(1)$ lookup time to find the next optimal arc (and hence the direction) but requires $O(n^2)$ storage for an n-node graph. This approach was shown to work well for small graphs, but is infeasible for large ones because of the excessive storage requirements. To remedy this problem Huang et al. [10] develop a hierarchical encoding structure, designed to reduce the space requirement at the expense of a modest increase in query time. They partition the graph into smaller subgraphs and then recursively construct a super-node structure on top of it. Different partitioning strategies are evaluated in [9]. Through simulations and empirical studies, they show that their path encoding outperforms the classic A* algorithm. Their results are mainly empirical, obtained by applying heuristic methods to limited datasets. The complexity bounds on their algorithms are not provided.

In [2] Agrawal et al. use a branch and bound search to reduce the total number of nodes visited. They partition the graph into "domains" (with identified centers) and precompute additional information to help prune the search space. However since ideal domain partitioning required is an NP-hard problem, they apply heuristics. Although they are able to reduce the search space for shortest path queries on large graphs.

Jung et al. [11] developed an algorithm based on hierarchical multi-graphs constructed by using boundary nodes and precomputed path information resulting from a partitioning of the graph into disjoint subgraphs. The shortest path algorithm used is a variation of the A^* algorithm which exploits this hierarchical structure. With simulations on synthetic data (grid graphs), the authors showed that HiTi performs better than A^*. However their worst case search space is still $O(V)$. Although this is an improvement over $O(|E|)$ of A^*, it is not a complexity improvement for transportation graphs since these graphs are sparse and in this case $|E| = O(V)$.

1.2 Outline of This Paper

The underlying digraph for Los Angeles county and its surrounding vicinity has about 177,748 nodes and 494,452 edges. Thus, it is most likely that a memory resident database would suffice for centralized ITS database servicing shortest path queries

in-degree	out-degree	%
2	1	0.9
1	2	0.9
2	2	1.1
1	1	19.1
4	4	20.9
3	3	55.3
rest		1.8

Table 1: Degree percentages of LA County graph.

for such a metropolitan area, and we consider only on such algorithms.

The overall goal of this paper may be stated briefly as follows. Let us assume a memory-resident[1] graph representing the transportation system for a large metropolitan area, such as Los Angeles County. At the extremes, shortest path queries could be answered on–the–fly by an algorithm such as Dijkstra's, for which the time complexity is $O(V \lg E)$ and space requirements are extremely modest. On the other hand, to minimize time, we might instead precompute shortest paths for all pairs of vertices, and reduce answering queries to a simple constant-time lookup. However, the latter approach incurs an $O(V^2)$ cost in the space required to store the precomputed table. What we seek is a range of intermediate algorithms, for which there is a trade-off between time and available memory, and which take advantage of the special properties of transportation graphs.

In §3 we give an overview of a general algorithm that provides the tradeoff described above, under the assumption of an algorithm for construction balanced partitions of this graph with small separators. §4 demonstrates a provably good partitioning algorithms of this sort. Finally, in §5 we step back and look at statistical properties of the graph at hand, and describe a significantly simpler algorithm that achieves these same bounds in practice.

2 Transportation Graphs

Transportation graphs are representations of the connectivity of city and inter-city streets, roads, freeways and highways. The geographic database underlying such a graph is often quite sizable, with a wealth of geometric, geographic and other data (often with a resolution down to just a few feet). However, the underlying transportation graph, which captures the essential topology of the routes together with relevant metric data, is significantly more concise. In this graph, nodes correspond to street intersections, freeway entries and exits, and so forth; edges are just the roads that connect them; costs may be derived from speed limits and distances, or may utilize statistical or measured values that capture actual congestion on roads.

A statistical analysis of this graph shows low in– and out–degrees, as expected; but contrary to a common assumption in the literature ([11, 14]), this digraph is not modeled well by a rectangular 2D grid. (See Table 2.) As expected, the graph is almost planar, where the planarity is violated by the existence of overpasses, underpasses and tunnels. Hence algorithms developed for planar graphs are not directly applicable. The existence of such features also leads to a high genus. However, as expected, these graphs tend to have a small crossing number under its natural planar embedding. For example the graph of LA County has about 5500 crossing edges which is only 3% percent of the total nodes. This motivates the extensions of some well known separator algorithms developed for the planar graphs [13] to graphs of low crossing number, and to apply them to problems on real transportation graphs.

3 Shortest Path Queries

To simplify the presentation, we initially assume that the underlying graph is a perfect grid, or sufficiently regular that it can be partitioned into equal sized subgraphs by removal of a small set of nodes. A subset of vertices S of an n-vertex graph $G = (V, E)$ is an *$f(n)$-separator* if removal of S partitions V into two disjoint sets A and B such that (1) $|A|, |B| \leq \delta n$ where $0 < \delta < 1$, *i.e.*, S δ-splits V (2) $|S| \leq f(n)$. (3) $(A \times B) \cap E = \emptyset$.

As noted earlier, what we seek is a class of algorithms which offer a trade-off between time and available memory for shortest path queries on transportation graphs. We focus on finding the length of the shortest path; To answer direction-based queries the identity of the next arc on the shortest path should also be stored with its cost.

3.1 Binary partitioning

Binary partitioning is a recursive application of bisection by removal of small separators. The resulting hierarchy can be viewed as a binary tree whose root is the entire graph. The algorithm uses this hierarchy to encode the partially computed shortest path information and then efficiently recover queried paths (for either direction– or path–based systems).

[1] The database of Thomas Brothers, Inc. was made available to the ISLA/IDA project at USC and is the basis of the statistical and empirical work reported here.

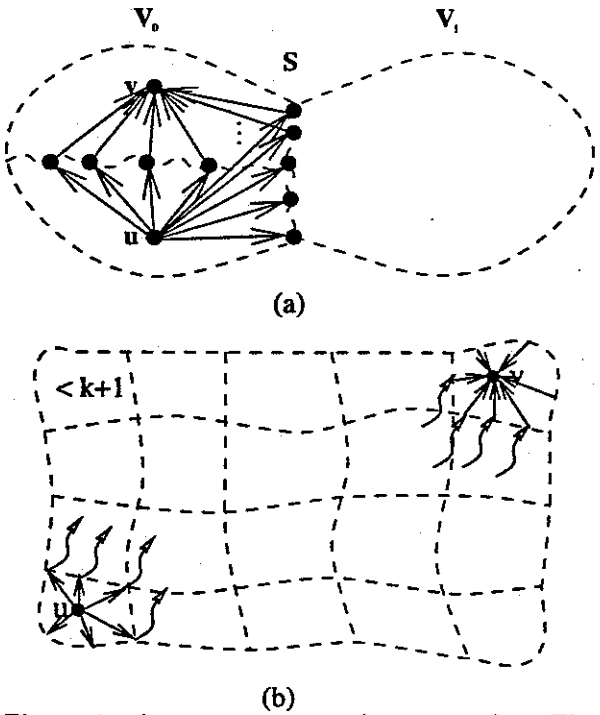


Figure 1: Arrows represent shortest paths. The distance associated to each arrow is retrieved by $O(1)$ lookup. The shortest path from u to v is the minimum of all the paths which pass through the separator sets. (a) Binary partitioning. (b) k-partitioning.

The idea is as follows: Partition a V vertex graph into two equal sized sub-graphs A and B by removing a small set of vertices (separator S). Next, precompute the shortest path paths between each pair of vertices in $A \times S$ and $B \times S$. Now suppose we want to find the shortest path from a vertex $u \in A$ to $v \in B$. Since every path from u to v has to include at least one vertex in S, it can be computed by taking the minimum of the lengths of all the shortest paths that include a vertex in S. The method can in turn be be applied recursively within each partition A and B to handle any source–destination pair. (See Figure 1 (a).)

The storage required by the data structures and time required to recover the shortest path is a function of the depth of the partition tree and the size of the separators at each level. Hence it is essential to minimize the size of the separators to optimize the performance of the algorithm.

3.2 k-Partitioning

A generalization of binary partitioning yields a k-partitioning algorithm that provides the desired space-time tradeoff. In k-partitioning, instead of partitioning the graph into two equal sized subgraphs we partition it into subgraphs of size (at most) k. As above, it is again desirable to have small separators associated with each partition. when the underlying graph is a perfect grid, and if we uniformly partition it into smaller grids of size $\sqrt{n}$ each subgraph will have at most $n^{1/4}$ separator nodes. The benefit will be a reduction the number of boundary nodes searched by a query algorithm at the expense of a more storage for precomputed data.

We construct the data structure as follows. We first choose a k and partition the graph. For each of the partition, we compute the shortest paths from (to) each node to (from) all the separator nodes around that partition. We also compute the all-pairs shortest path between all nodes in the separator. Within each partition, we reapply the k-partitioning algorithm, for a $k' < k$; however, our analysis and implementation, we merely utilize binary partitioning from this point on. Now suppose we are given a source node u and a destination node v. (Figure 1 (b)). We can find the length of the shortest path from u and v by taking the minimum distance among all the paths that include a separator node in u's partition and a separator node in v's partition, as above. Again the total size of the separator and their sizes relative to the size of each partition, are critical values which determine the complexity of the resulting algorithm.

4 A Partitioning Theorem

Recursive application of separator theorems leads to divide-and-conquer algorithms for a variety of applications. For most families of graphs, non-trivial separators do not exist. On the other hand, Lipton and Tarjan [13] proved planar graphs have separator set of size $\sqrt{8n}$ that yield $\frac{2}{3}$-splits. An extension, proved in [5] proves that graphs with genus g have $O(\sqrt{gn})$ separators that give similar splits. Similar results were also proved for other classes of graphs. Applications of graph separator theorems exist in VLSI layout and graph partition for finite element methods. and for various geometric problems.

To construct the appropriate partitions for transportation graphs, we require an extension of the Lipton–Tarjan theorem to graphs of low crossing number. The methods used are similar to ones developed by Leighton in [12].

Theorem 4.1 *Let $G = (V, E)$ be an n-vertex graph with crossing number $cr(G) = m$ and with nonnegative vertex costs summing no more than one. Then V can be partitioned into three disjoint sets V_0, V_1 and S such that no edge joins a vertex*

in V_0 to a vertex in V_1, neither V_0 nor V_1 has total cost exceeding 2/3, and S contains no more than $4\sqrt{2}\sqrt{V+m}$ vertices. Moreover, such a partition can be found in $O(V+m)$ time.

Note that when the crossing number is $O(V)$, the size of the separator is $O(\sqrt{V})$, and it can be found in $O(V)$ time, differing from the planar case only in the constant factor.

Applying Theorem 4.1 to the algorithms of §3, we can solve the direction–based query problem with preprocessing time $O(n\sqrt{n}\log n)$, space $O(n\sqrt{n})$, and query time complexities of $O(\sqrt{n})$ with binary partitioning; and, with k-partitioning, with preprocessing time $O(n^2\log n/\sqrt{k} + n\sqrt{k}\log k)$, space $O(n^2/k + n\sqrt{k})$, and query time $O(k)$, for $1 \le k \le n^{2/3}$. See [3] for details.

5 Applications

The goal of the work sketched in the previous section was to develop an algorithm that to be deployed a component of a large scale GIS database currently under development by the ISLA/IDA project at the University of Southern California. The initial plan was to implement the algorithm as described above; however, a bit of research revealed that, although planar separators have been discussed in a variety of contexts, to our knowledge the algorithm has rarely (if ever) been implemented. In practice, simpler algorithms may indeed suffice to achieve comparable results. Such is a case in our situation, since transportation graphs have some unique characteristics that can be exploited. The algorithm we used is suggested by observed properties of the graph at hand. In addition, we wanted a partition that would simplify planar point location within the region and would mesh well with data retrieval from the underlying database.

A straightforward approach might be to partition the graph along a uniform grid, hoping that all of the cells will contain nearly the same number of nodes. This approach may work if the graph is of uniform density. However this property is easily violated on large, non-uniform maps like that of the Los Angeles environs, which ranges from a dense downtown area to residential regions to hills and mountains. (Figure 2 attempts to quantify this in a histogram of nodes per cell for uniform grid partitions of various resolutions on the map of LA. For clarity, cells with no nodes in it are omitted; these constitute 77% of the entire region.)

However, while the density of the embedding is not uniform, it does exhibit a sort of local regularity; specifically, the graph itself is almost planar (with a very low crossing number, and crossing edges well distributed through the graph); and the degrees of the vertices are bounded. In addition, we would also like to know that the distribution of nodes and edges is reasonably uniform, in the sense that if when we choose an sufficiently large convex region, the number of edges crossing its boundary is $O(\sqrt{n})$, where n is the number of nodes contained in the region. Fortunately, these graphs arise from transportation systems, which are engineered to maximize accessibility, so this property seems to hold in general.[2]. These properties motivated us to construct a partition of the graph in a manner similar to the construction of an *R-tree* [6] — explicitly attempting to construct balanced partitions via horizontal and vertical separators, while counting on these observed properties of the graph to bound the size of the resulting separators.

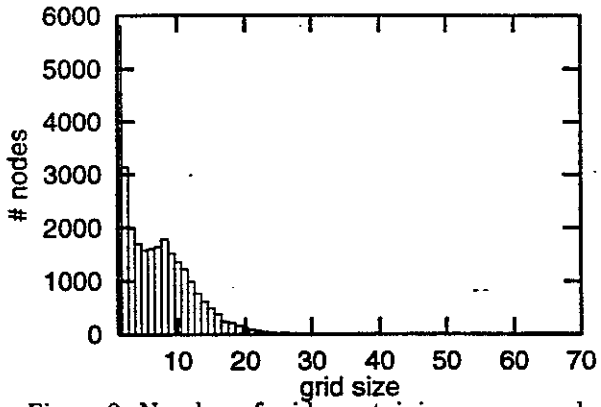


Figure 2: Number of grids containing same number of nodes.

The simple algorithm we employ partitions regions using either horizontal or vertical lines: points are sorted by their x and y coordinates. A line is drawn to split the region into two balanced partitions, reducing the size of the bounding box of each resulting region along the maximum dimension of the enclosing bounding box. The process is then iterated recursively, yielding an overall time of $O(V \lg V)$ for the entire partition. Such a strategy is expected to give a balanced partitioning although

[2]We tested this property with the following experiment on the abovementioned LA County transportation graph (75 x 105 square miles). If the underlying graph were a perfect grid and we partitioned the map into a regular grid, then the number of boundary edges for each grid cell is expected to be about 4 times the square root of the number of nodes in each partition. We constructed a range of regular rectangular partitions of the graph, and found the real number of edges crossing the boundary of each; we divided this by the square root of the number of nodes actually in the partition and averaged the value over similarly sized regions covering the entire map. We repeated this for regions of size $2^i/4 \times 2^i/4$ square miles for $i = 0, \cdots, 7$. The results are shown in Figure 3.

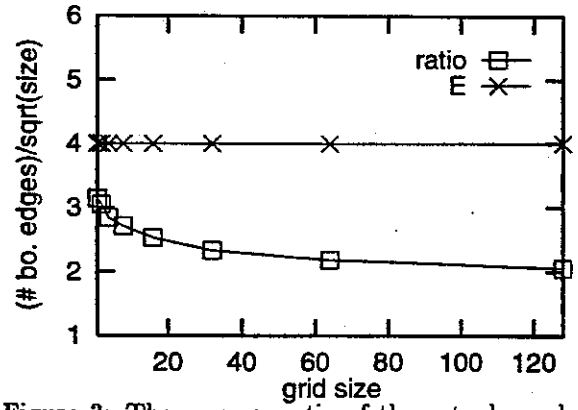


Figure 3: The average ratio of the actual number of edges crossing a region's boundary to the square root of the number of nodes in that region for a uniform grid partition of the LA County transportation map, for grids of varying density. The value for a perfect grid is E = 4.

there is no guarantee that the induced separator sets will be small for each partition.

Based on the promising results of Figure 3, we replaced the complex algorithm derived from the Separator Theorem with the heuristic R-tree algorithm. Because of space limitations, we discuss results only for k-partitioning, for various values of k. To illustrate the goodness of the result, we computed the ratio of the number of actual separator nodes to the square root of the number of nodes in the partition and plotted the results for $k = 80, 160, 320$ in Figure 4. The line $x = 4$ is the value corresponding to a perfect grid. We can easily conclude that the practical algorithm provides an excellent partitioning of real transportation graphs. The total storage required by the constructed data structures for $k = 320$, for example, requires about 45 megabytes, which makes a memory resident solution feasible for an ITS center. We can also conclude that because of the flexibility of choosing an optimized k for an existing hardware, the k-partitioning algorithm appears to be better than the binary partitioning.

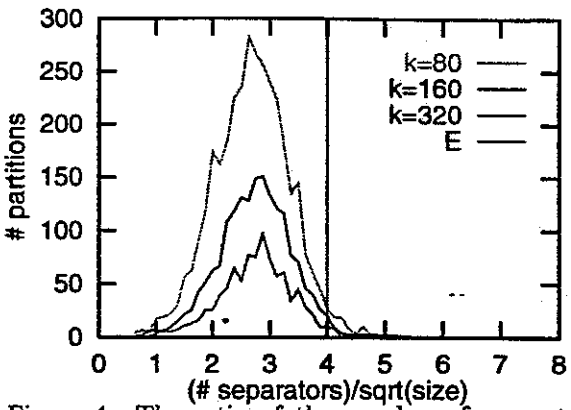


Figure 4: The ratio of the number of separator nodes the square root of the partition size as k changes.

References

[1] R. Agrawal and H. V. Jagadish. Materialization and incremental update of path information. In *Proceedings of the 5th International Conference on Data Engineering*, pages 374–383, 1989.

[2] R. Agrawal and H. V. Jagadish. Algorithms for searching massive graphs. *Transactions on Knowledge and Data Engineering*, 6(2):225–238, April 1994.

[3] C. Aydin and D. Ierardi. Partitioning real transportation graphs with applications to routing. Technical report, University of Southern California, 1997.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1993.

[5] J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan. A separator theorem for graphs of bounded genus. *Journal of Algorithms*, 5:391–407, 1984.

[6] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conference*, pages 47–57, 1984.

[7] M. A. W. Houtsma, F. Cacace, and S. Ceri. Parallel hierarchical evaluation of transitive closure queries. In *Proceedings of the 1st International Conference on Parallel and Distributed Information Systems*, pages 130–137, 1990.

[8] Y.-W. Huang, N. Jing, and E. A. Rundensteiner. A semi-materialized view approach for route maintenance in ivhs. In *Proceedings of the 2nd ACM Workshop on Geographic Information Systems*, pages 144–151, 1994.

[9] Y.-W. Huang, N. Jing, and E. A. Rundensteiner. Effective graph clustering for path queries in digital map databases. In *Proceedings of the 5th International Conference on Information and Knowledge Management*, 1996.

[10] Y.-W. Huang, N. Jing, and E. A. Rundensteiner. Hierarchical optimization of optimal path finding for transportation applications. In *Proceedings of the 5th International Conference on Information and Knowledge Management*, 1996.

[11] S. Jung and S. Pramanik. Hiti graph model of topological road maps in navigation systems. In *Proceedings of the 12th International Conference on Data Engineering*, pages 76–84, 1996.

[12] F. T. Leighton. *Complexity Issues in VLSI*. Foundations of Computing. The MIT Press, Cambridge, MA, 1983.

[13] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math*, 36:177–189, 1979.

[14] S. Shekar, A. Kohli, and M. Coyle. Path computation algorithms for advanced traveler information systems. In *Proceedings of the 9th International Conference on Data Engineering*, pages 31–39, 1993.

[15] Loral Federal Systems, IVHS architecture phase one final report. Sponsored by Federal Highway Administration, DTFH61-93-C-00211, 1994.

Stability of Voronoi Neighborship under Perturbations of the Sites

Frank Weller

ABSTRACT

This paper considers the effect of site perturbations on Voronoi diagrams, where the sites are points in the plane. Given a bound on the distance that any site may move, we ask which pairs of Voronoi neighbors may become non-neighbors and which are guaranteed to remain neighbors. A pair of the second kind is called stable.

The paper shows necessary and sufficient conditions for stability. Algorithms are proposed for deciding stability with regard to a given perturbation bound and for determining the supremal bound up to which a particular pair of Voronoi neighbors remains stable.

1. INTRODUCTION

The Voronoi diagram of a set of points is a powerful tool for proximity-related computations. It is used by many algorithms in computational geometry and related fields. When dealing with real-world data, errors of measurement can have a non-negligible impact on the result. In the case of Voronoi diagrams, even very slight perturbations of the sites may change the diagram's topology. If an algorithm makes decisions based on a Voronoi diagram, it may benefit from the knowledge how reliable the diagram as a whole or certain parts of it are. The stability which we investigate here can serve as a measure of reliability.

This paper is concerned with the effect that site perturbations have on the topology of a Voronoi diagram in the plane. In particular, we want to know which pairs of strong Voronoi neighbors are 'separated' by the perturbation and which are not. It is assumed that the reader is familiar with planar Voronoi diagrams. An introduction can be found in, e.g., [PS85], [Ede87], and [Aur91].

For the sake of discussion, we assume that we have an 'exact' and a 'measured' set of points P and P', respectively, in the plane. The error of measurement is bounded above by $\varepsilon > 0$. I.e., the distance $|p_i p_i'|$ between an 'exact' site p_i and its 'measured' counterpart p_i' satisfies $|p_i p_i'| \leq \varepsilon$. All sites are allowed to move simultaneously, and the same bound holds for all sites. Obviously, the Voronoi diagrams V of P and V' of P' may differ in their topologies. However, for certain pairs of Voronoi neighbors in V one can guarantee that their 'measured' counterparts will also be Voronoi neighbors in V'. We call such a pair ε-stable. Section 2 defines ε-stability and establishes a test criterion. It turns out that the stability of a neighbor pair depends only on the Voronoi neighbors of this pair.

Two computational problems arise in this context. The decision problem asks whether a pair of neighbors is stable for a given value of ε. The optimization problem looks for the supremal ε up to which a given pair of neighbors remains stable. Algorithms for both problems are developed in Section 3. These algorithms inspect only the strong Voronoi neighbors of the two sites under consideration. Their worst-case running time is linear in the number of inspected neighbors.

The stability of various geometric graphs under site perturbation has been investigated by Abellanas et al. [AGH+93]. They use the same model of perturbation which is presented here, but their graphs are not based on proximity nor, in fact, on any metric.

2. STABILITY OF VORONOI NEIGHBORSHIP

This section starts with a formal definition of ε-stability. Necessary and sufficient conditions for ε-stability of a pair of Voronoi neighbors are established.

Supported by Deutsche Forschungsgemeinschaft (DFG), grant MU 744/3-2.

Let $P = \{p_1, \ldots, p_n\}$ be the set of unperturbed sites and $P' = \{p'_1, \ldots, p'_n\}$ the perturbed set. Euclidean distance between points p and q is denoted by $|pq|$. We call P' an *ε-perturbation* of P if $|p_i p'_i| \leq \varepsilon$ holds for all i, $1 \leq i \leq n$.

The type of neighborship which we consider is *strong Voronoi neighborship*. Two sites are strong neighbors if their Voronoi regions share an edge. Weak neighborship, in contrast, means that the Voronoi regions share only a vertex. A pair of strong Voronoi neighbors $p_i, p_j \in P$ is called *ε-stable* if their counterparts p'_i and p'_j are strong Voronoi neighbors for all ε-perturbations P'. If ε is chosen large enough, it becomes possible to move two sites of P onto each other, such that $p'_i = p'_j$ for $i \neq j$. In this case, we define the neighborship of p_i or p_j with any other site to be ε-unstable.

Strong Voronoi neighborhood is defined by the existence of a non-degenerate Voronoi edge between the two sites. This is equivalent to the existence of interior points of the Voronoi edge. The following lemma gives a characterization in slightly weaker terms, which we shall need later on.

Lemma 1. *Sites $p_i, p_j \in P$ are strong Voronoi neighbors if and only if there exists a point m such that*

$$\max\{|mp_i|, |mp_j|\} < \min_{k \neq i,j} |mp_k| \quad . \tag{1}$$

Proof. Let m satisfy (1) and, w.l.o.g., let $|mp_i| \geq |mp_j|$. The line segment mp_i intersects the perpendicular bisector g of p_i and p_j in exactly one point m'. We have

$$\begin{aligned} \max\{|m'p_i|, |m'p_j|\} &= \max\{|mp_i|, |mp_j|\} - |mm'| \\ &< \min_{k \neq i,j} |mp_k| - |mm'| \\ &\leq \min_{k \neq i,j} |m'p_k| \quad , \end{aligned}$$

so m' satisfies (1) and is equidistant from p_i and p_j. This means that m' is an interior point of the Voronoi edge v. In particular, v exists and is non-degenerate.

On the other hand, (1) holds for any interior point of v. □

The point m is the center of a circular disk D containing p_i and p_j, but no further site of P.

The perturbation bound allows each p'_i to move within a circular disk of radius ε centered at p_i. We denote this ε-disk by D_i. The condition for ε-stability is very similar to that in Lemma 1. There must exist a disk D that contains D_i and D_j but does not intersect any other ε-disk of P. The following lemma formulates this in terms of the center m of D.

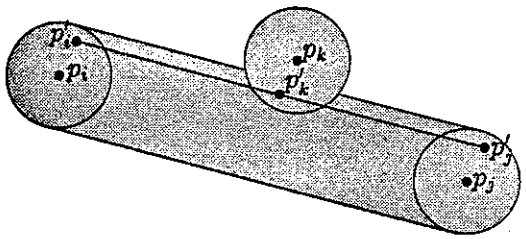


FIGURE 1. The convex hull of ε-disks D_i and D_j is intersected by another ε-disk D_k.

Lemma 2. *A pair of Voronoi neighbors $p_i, p_j \in P$ is ε-stable if and only if $|p_i p_j| > 2\varepsilon$ and there exists a point m such that*

$$|mp_i| + \varepsilon = |mp_j| + \varepsilon < \min_{k \neq i,j} |mp_k| - \varepsilon \quad . \tag{2}$$

Proof. If $|p_i p_j| \leq 2\varepsilon$, then $p'_i = p'_j$ for some P', so the neighborship is unstable by definition. For the rest of the proof, assume that $|p_i p_j| > 2\varepsilon$.

Let (2) hold for point m. For any ε-perturbation P', we have

$$\begin{aligned} \max\{|mp'_i|, |mp'_j|\} &\leq \max\{|mp_i|, |mp_j|\} + \varepsilon \\ &< \min_{k \neq i,j} |mp_k| - \varepsilon \\ &\leq \min_{k \neq i,j} |mp'_k| \end{aligned}$$

due to the triangle inequality. By Lemma 1, p'_i and p'_j are strong Voronoi neighbors within P'.

For the converse, let $p_i p_j$ be ε-stable. We note that no ε-disk D_k, $k \neq i, j$, intersects the convex hull of $D_i \cup D_j$. If an intersection existed, we could find an ε-perturbation such that p'_k lies on the line segment $p'_i p'_j$ (see Figure 1.) With this perturbation, p'_i and p'_j cannot be Voronoi neighbors.

Since p_i and p_j are ε-stable neighbors for some $\varepsilon > 0$, they are strong Voronoi neighbors. There exists a point m satisfying (1). W.l.o.g., m lies on the bisector g of p_i and p_j. Let C be the circle through p_i and p_j centered at m. We apply two transformations to C. After these transformations, the new center m will satisfy (2).

The first transformation modifies the radius of C, leaving the center m unchanged. Since all p_k, $k \neq i, j$, lie strictly exterior to C, we can find a radius such that

(1) C touches ε-disk D_ℓ for some $\ell \neq i, j$,
(2) the interior of C intersects no D_k for $k \neq i, j$, and
(3) D_i and D_j intersect the interior of C.

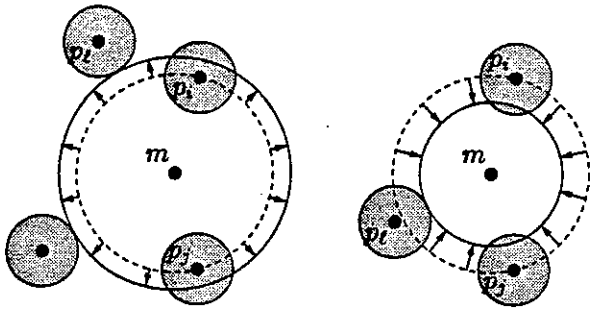


FIGURE 2. The first transformation changes the radius of C.

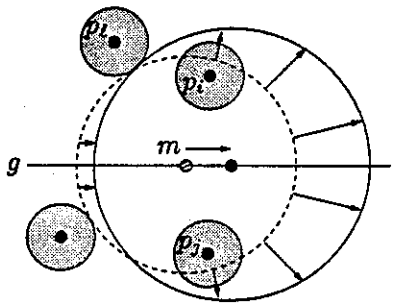


FIGURE 3. The second transformation moves the center m of C.

Figure 2 shows two examples of the first transformation. Note that C may expand or shrink in this step.

If D_i and D_j lie strictly interior to C after the first transformation, we are finished, since m satisfies (2). Otherwise, both D_i and D_j intersect C, and $C \setminus (D_i \cup D_j)$ has two connected components. W.l.o.g., let the bisector g of p_i and p_j lie horizontally and let D_ℓ touch the left component of $C \setminus (D_i \cup D_j)$, as in Figure 3. If more than one ε-disk touches the left component, let D_ℓ be one that lies rightmost.

The second transformation is shown in Figure 3. The center m of C moves to the right along the bisector g of p_i and p_j. As m moves, we adjust the radius of C continuously such that C keeps touching D_ℓ. We stop at some point where the interior of C contains both D_i and D_j but does not yet intersect any other ε-disk. Since D_ℓ does not intersect the convex hull of $D_i \cup D_j$, we will always find a circle that touches D_ℓ and has D_i and D_j in its interior. However, it may be impossible to transform to this circle, as an intersection with some other ε-disk D_k may occur on the way. If D_k intersects the right component of $C \setminus (D_i \cup D_j)$, then we can move p'_i, p'_j, p'_ℓ, and p'_k onto C, as shown in Figure 4. This creates an ε-perturbation in which p'_i and p'_j are only

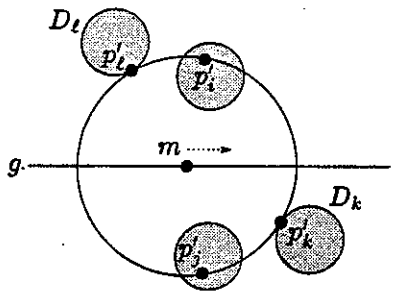


FIGURE 4. If the second transformation is stopped by some ε-disk on the right, edge p_ip_j is unstable.

weak neighbors, in contradiction to our assumption that p_i and p_j are ε-stable. Therefore, D_k must intersect the left component of $C \setminus (D_i \cup D_j)$. From the way C changes under the second transformation (compare Figure 3,) it is easy to see that D_k must lie further to the right than D_ℓ. We continue to move m to the right, but now C keeps touching D_k instead of D_ℓ. At the end of the second transformation, m satisfies (2). □

Remark 3. *The equation in* (2) *restricts m to lie on the perpendicular bisector g of points p_i and p_j. The inequation in* (2) *and the fact that $\varepsilon > 0$ further restrict m to the Voronoi edge $v \subseteq g$ of p_i and p_j.*

Lemma 4. *Let $p_i, p_j \in P$ be strong Voronoi neighbors, and let m be a point of their common Voronoi edge. Then*

$$\min_{p_k \in P, k \neq i,j} |mp_k| = \min_{p_s \in S, s \neq i,j} |mp_s| \quad ,$$

where $S \subseteq P$ contains the strong Voronoi neighbors of p_i and the strong Voronoi neighbors of p_j.

Proof. Let p_ℓ be neither a Voronoi neighbor of p_i nor of p_j. The line segment mp_ℓ intersects the closed Voronoi region of some site p_s such that p_s is a strong Voronoi neighbor of p_i or of p_j. Let q be a point of intersection with that Voronoi region, then

$$|p_sq| \leq |p_\ell q| \quad .$$

By the triangle inequality applied to p_s, q and m, and by choice of q, we have

$$|p_sm| \leq |p_sq| + |qm| \leq |p_\ell q| + |qm| = |p_\ell m| \quad .$$

We see that site p_ℓ has no influence on $\min_{k \neq i,j} |mp_k|$. This is true even when p_ℓ and p_i (or p_j, respectively) are weak Voronoi neighbors. □

Remark 3 and Lemma 4 allow us to weaken the condition of Lemma 2 to

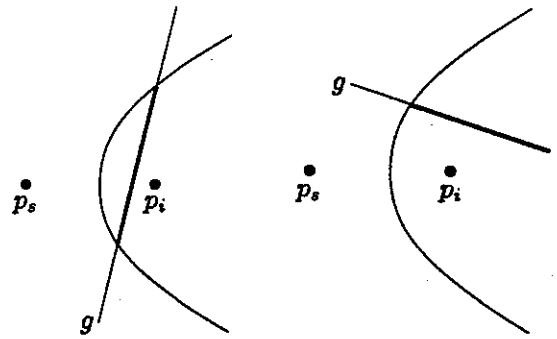


FIGURE 5. Possible non-empty intersections of bisector g with the convex hyperbolic region R_{si}.

Theorem 5. *A pair of Voronoi neighbors $p_i, p_j \in P$ is ε-stable if and only if $|p_ip_j| > 2\varepsilon$ and there exists a point $m \in v$ such that*

$$(3)\quad |mp_i| + \varepsilon = |mp_j| + \varepsilon < \min_{p_s \in S, s \neq i,j} |mp_s| - \varepsilon \quad ,$$

where $S \subseteq P$ contains the strong Voronoi neighbors of p_i and the strong Voronoi neighbors of p_j.

3. ALGORITHMS

With Theorem 5 in mind, it is relatively easy to decide stability for a given $\varepsilon > 0$. After first checking that $|p_ip_j| > 2\varepsilon$, the algorithm tries to identify points which satisfy (3).

Let us first consider only p_i and one other site p_s, $s \neq i, j$. If $|p_sp_i| < 2\varepsilon$, edge p_ip_j is unstable. Otherwise, let region R_{si} consist of those points m of the plane which satisfy

$$(4)\qquad \begin{aligned} & |mp_i| + \varepsilon < |mp_s| - \varepsilon \\ \Leftrightarrow \quad & |mp_s| - |mp_i| > 2\varepsilon \quad . \end{aligned}$$

R_{si} is convex. Its boundary is the hyperbolic branch H_{si} defined by

$$(5)\qquad |mp_s| - |mp_i| = 2\varepsilon \quad .$$

Since $|p_ip_s| > 2\varepsilon$, the hyperbola does not degenerate. In particular, it always exists.

Now consider the bisector g of p_i and p_j, and let I_s be the subset of g where (4) holds. We find I_s by intersecting g with H_{si}. I_s is either empty, an open line segment, or an open half-line. The latter two cases are shown in Figure 5. Note that

$$g \cap R_{si} = g \cap R_{sj} \quad ,$$

since g is the bisector of p_i and p_j. Thus, it makes no difference whether we use p_i or p_j to compute I_s.

Our decision algorithm computes I_s for each strong Voronoi neighbor $p_s \in S$. The intersection of these I_s is exactly the set of points where (3) holds.

Algorithm 1.

1. If the distance of p_i or p_j to its respective nearest neighbor is $\leq 2\varepsilon$, then edge p_ip_j is ε-unstable.
2. For each Voronoi neighbor p_s of p_i or p_j, $s \neq i, j$, compute the open segment $I_s \subseteq g$.
3. Intersect all I_s.
4. The neighborship of p_i and p_j is ε-stable if and only if the intersection is non-empty.

The algorithm examines $\#S - 2$ neighbor sites p_s. They can be found from the Voronoi diagram in time $O(\#S)$, identifying at the same time the nearest neighbors for Step 1. Intersecting R_{si} and g is basically solving a quadratic equation, which takes constant time. Each I_s is represented as an open interval on the real number line. The intersection of $\#S - 2$ intervals takes $O(\#S)$ time. The total complexity of the decision algorithm is $O(\#S)$. This can still be $O(n)$ in the worst case. However, the average number of strong Voronoi neighbors of a single site is well-known to be less than 6.

The optimization algorithm is given a pair of strong Voronoi neighbors p_i and p_j. It determines the value

$$\varepsilon_{\sup} := \sup \{\varepsilon : p_i \text{ and } p_j \text{ are } \varepsilon\text{-stable neighbors}\}$$

First, we note that p_ip_j is unstable when the ε-disks D_i and D_j intersect. Therefore, we have $\varepsilon_{\sup} \leq \frac{1}{2}|p_ip_j|$. Our algorithm will contain an explicit test to ensure that the computed result will not violate this bound.

For all $\varepsilon < \varepsilon_{\sup}$ there exists a point m on the Voronoi edge v of p_i and p_j such that m satisfies (3). Therefore, $2\varepsilon_{\sup}$ equals the supremal value of function

$$f^{ij}_{\min}(m) := \min_{P_s \in S, s \neq i,j} |mp_s| - |mp_i|$$

(unless this value is greater than $|p_ip_j|$.) For each neighbor site p_s, $s \neq i, j$, we define the function

$$f_{si}(m) := |mp_s| - |mp_i| \quad .$$

Obviously, $f^{ij}_{\min}$ is the pointwise minimum of all f_{si}.

For the actual computation of $\varepsilon_{\sup}$, we choose a coordinate system with g as the x-axis and p_i on the positive y-axis. Each p_s is represented by coordinates (x_s, y_s). The functions f_{si} are now of the form

$$f_{si}(m) =: f_{si}(x) = \sqrt{(x - x_s)^2 + y_s^2} - \sqrt{x^2 + y_i^2} \quad .$$

Function f_{si}, which is defined only on $v \subseteq g$, does not change if we reflect p_s across g. Therefore, we may assume w.l.o.g. that $y_s \geq 0$ for all $s \neq j$. Our algorithm

will change the sign of y_s if necessary. Should the set P contain two sites $p_s = (x_s, y_s)$ and $p_t = (x_s, -y_s)$, then $f_{si} = f_{ti}$ and we can discard one of the functions.

Examining $\#S - 2$ neighbor sites of p_i and p_j, we determine the supremum of $f^{ij}_{\min}$ over g in time $O(\#S)$. In order to do so, we modify Megiddo's algorithm for linear programming in the plane [Meg83, Section 2]. This algorithm maximizes (or, in the original formulation, minimizes) y within a planar region. The region is defined by a feasible interval $[a, b]$ of the x-axis, a set of upper constraints, and a set of lower constraints. The feasible interval may be unbounded, in which case one sets $a = -\infty$ and/or $b = \infty$. For ease of discussion, we retain the notation $[a, b]$ in these cases. In Megiddo's original algorithm, each constraint is given as a linear function of x. In our case, the upper constraints are the functions $f_{si}(x)$, and there are no lower constraints. Our feasible interval is the Voronoi edge v of p_i and p_j. By maximizing y under the constraint functions f_{si}, we find the supremum of $f^{ij}_{\min}$.

Upon close inspection, it turns out that Megiddo's method is not limited to linear constraints. It can also handle a set of upper constraints satisfying the following conditions:

U1. Let constraint function f assume its maximum over $[a, b]$ at $x_{\max}$, then f is non-decreasing to the left of $x_{\max}$ and non-increasing to the right of $x_{\max}$. The point $x_{\max}$ can be found in constant time.

U2. For any constraint f and any $x \in [a, b]$, $f(x)$ can be evaluated in constant time.

U3. For any pair of constraint functions f and h, it can be determined in constant time whether $f(x) \geq h(x)\ \forall x$ or $h(x) \geq f(x)\ \forall x$ holds. If neither is true, then the graphs of f and h cross at most once and the crossing point can be computed in constant time.

In the formulation of these conditions, we consider the domain $[a, b]$ of the constraint functions to be a closed, possibly infinite interval of the closed real line $\mathbf{R} \cup \{-\infty, \infty\}$. E.g., an asymptotic supremum for $x \to \infty$ is simply considered as a maximum at $x = \infty$.

Certain decisions in Megiddo's algorithm are based on whether a constraint is increasing or decreasing at some x. Others depend on which of two constraints is smaller to the left of their crossing point, and which is smaller to the right. With linear constraints, these tests are done by examining the constraints' slopes. Conditions U1–U3 allow us carry out the tests in constant time without computing derivatives.

In addition to Conditions U1–U3, Megiddo's algorithm exploits the fact that a linear constraint function always assumes its maximum at $x = a$ or $x = b$. The modification that is necessary to accomodate for maxima over the interior of $[a, b]$ is straightforward.

It remains to be shown that our set of constraint functions satisfies Conditions U1–U3. The triangle inequality implies that f_{si} is bounded above and below by $|p_s p_i|$ and $-|p_s p_i|$, respectively. One of the bounding values is assumed in the intersection point of line $p_i p_s$ with g, provided that the intersection exists. The intersection occurs at

$$x_0 = \frac{-y_i x_s}{y_s - y_i} \quad . \tag{6}$$

No intersection exists if $y_s = y_i$.

Lemma 6. *Condition* U2 *holds for all* f_{si}, $s \neq i, j$.

Proof. It is clear that our constraint functions can be evaluated in constant time for $x \in \mathbf{R}$. Elementary calculus yields

$$\lim_{x \to -\infty} f_{si}(x) = x_s \quad \text{and} \quad \lim_{x \to \infty} f_{si}(x) = -x_s \quad ,$$

which is of importance for the cases $a = -\infty$ and $b = \infty$. □

Lemma 7. *Condition* U1 *holds for all* f_{si}, $s \neq i, j$.

Proof. The derivative of f_{si} w.r.t. x is

$$\begin{aligned} f'_{si}(x) &= \frac{2(x - x_s)}{2\sqrt{(x - x_s)^2 + y_s^2}} - \frac{2x}{2\sqrt{x^2 + y_i^2}} \\ &= \frac{(x - x_s)\sqrt{x^2 + y_i^2} - x\sqrt{(x - x_s)^2 + y_s^2}}{\sqrt{(x - x_s)^2 + y_s^2}\sqrt{x^2 + y_i^2}} \quad . \end{aligned}$$

As y_i is non-zero, the denominator of f'_{si} can only vanish if $y_s = 0$ and $x = x_s$. This means that $p_s \in g$ and $m = p_s$. Now clearly $m = p_s$ does not belong to the Voronoi edge v, so $x = x_s$ lies outside the feasible interval $[a, b]$. For all other combinations of x and y_s, function f_{si} is continuously differentiable. A necessary condition for a local extremum of f_{si} over the open interval (a, b) is

$$\begin{aligned} & x^2(x - x_s)^2 + y_i^2(x - x_s)^2 = x^2(x - x_s)^2 + x^2 y_s^2 \\ \Leftrightarrow \quad & y_i^2(x - x_s)^2 = x^2 y_s^2 \\ \Leftrightarrow \quad & 0 = (y_s^2 - y_i^2)x^2 + 2x_s y_i^2 x - y_i^2 x_s^2 \\ & = \big((y_s - y_i)x + y_i x_s\big)\big((y_s + y_i)x - y_i x_s\big) \\ & =: h(x) \quad . \end{aligned}$$

Consider the linear factors in the last-but-one line of this equation. Since $y_i > 0$ and $y_s \geq 0$, the leading coefficient $(y_s + y_i)$ of the second linear factor is not equal to 0. We distinguish two cases:

$y_s \neq y_i$:
$h(x)$ has two roots,

$$x_1 = \frac{-y_i x_s}{y_s - y_i} \quad \text{and} \quad x_2 = \frac{y_i x_s}{y_s + y_i} \quad .$$

$y_s = y_i$:
In this case we have $x_s \neq 0$, for otherwise p_s would equal p_i. The first linear factor of $h(x)$ becomes a non-zero constant, and we obtain only one root,

$$x_2 = \frac{y_i x_s}{y_s + y_i} \quad . \tag{7}$$

We have encountered x_1 before, under the name of x_0 in (6). Point $(x_1, 0)$ is the intersection of line $p_i p_s$ with g, and $f_{si}(x_1) = \pm |p_s p_i|$. Since $[a, b]$ describes the Voronoi edge of p_i and p_j, it is exactly the interval over which none of the f_{si} is negative. If $x_1 \in [a, b]$, then $f_{si}(x_1) = |p_s p_i|$ and f_{si} has a global maximum at x_1.

The other root, x_2, corresponds to the intersection point of line $p_j p_s$ with g. If $x_s = 0$, then x_2 coincides with $x_1 = 0$. If $x_s \neq 0$, lines $p_s p_j$ and g intersect at a non-right angle. This implies that $|mp_j| = |mp_i|$ increases for $m \in g$ and m on one side of the intersection, and decreases on the other side. The behavior of $|mp_s|$ is opposite to that of $|mp_i|$. (At the extreme, $p_s \in g$ and $|mp_s|$ is stationary in the intersection point.) Thus, f_{si} has no local extremum at x_2.

At this point, x_1 is the only candidate point at which a local extremum within (a, b) might occur, and this extremum must be a maximum. If x_1 exists and $x_1 \in (a, b)$, then $x_{\max} = x_1$. Otherwise, we have $x_{\max} = a$ or $x_{\max} = b$, depending on the values $f_{si}(a)$ and $f_{si}(b)$. We see that $x_{\max}$ can be found in constant time. □

Finally, we need to verify that the graphs of any two constraint functions cross at most once.

Lemma 8. U3 *holds for all pairs of constraint functions* f_{si}, $f_{\ell i}$.

Proof. Consider the difference function

$$\begin{aligned} f_{si}(m) - f_{\ell i}(m) &= |mp_s| - |mp_i| - (|mp_\ell| - |mp_i|) \\ &= |mp_s| - |mp_\ell| \quad . \end{aligned}$$

The roots of this function lie in the intersection of g with the bisector of p_s and p_ℓ. If $x_s = x_\ell$, then the bisector is parallel to g. Now if $y_s \geq y_\ell$ then $f_{si} \geq f_{\ell i}$, and if $y_s \leq y_\ell$ then $f_{si} \leq f_{\ell i}$ over all of v. If $x_s \neq x_\ell$, then the intersection consists of exactly one point, $(x_3, 0)$, say. Since g crosses the bisector of p_s and p_ℓ in this point, we see that the graphs of f_{si} and $f_{\ell i}$ cross at x_3. □

The following algorithm determines ε_{sup}.

Algorithm 2.

1. Transform the coordinates of all Voronoi neighbors of p_i and of p_j into the xy system defined by g. Reflect the neighbors across g if necessary.
2. Compute the supremum of $f^{ij}_{\min}$.
3. Set ε_{sup} to one half of the supremum or $\frac{1}{2}|p_i p_j|$, whichever is smaller.

The first two steps of the algorithm take time $O(\#S)$. The third step takes only constant time, resulting in worst-case time $O(\#S)$ for the complete algorithm.

4. Conclusion

We have examined the stability of strong Voronoi neighborship under ε-perturbation of the sites. Stability of two neighbors p_i and p_j is characterized by the existence of a circle which includes the ε-disks around p_i and p_j and excludes the ε-disks around all other sites. Such a circle can be found in time $O(\#S)$, where $\#S - 2$ is the number of strong Voronoi neighbors of p_i and p_j. Likewise, the supremum of all values ε for which p_i and p_j are stable neighbors can be computed in time $O(\#S)$. As a by-product, we have seen that Megiddo's approach to linear programming in the plane is applicable to a wider range of problems with more general classes of constraint functions.

References

[AGH+93] M. Abellanas, J. García, G. Hernández, F. Hurtado, O. Serra, and J. Urrutia. Updating polygonizations. *Computer Graphics Forum*, 12(3):C134–C152, 1993.

[Aur91] Franz Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, September 1991.

[Ede87] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. EATCS Monographs on Theoretical Computer Science 10. Springer, Berlin, 1987.

[Meg83] Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, November 1983.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry*. Springer, 1985.

An iterative algorithm for the determination of Voronoi vertices in polygonal and non-polygonal domains

François Anton, and Christopher Gold
Industrial Chair of Geomatics - CRG
0722 Casault, Université Laval
Sainte-Foy, Québec, Canada, G1K 7P4
Fax: (+1-418) 656-7411
E-mail: francois@gmt.ulaval.ca & Christopher.Gold@scg.ulaval.ca

Abstract

We propose a new iterative algorithm for the computation of the vertices of a Voronoi diagram for a set of geometric objects of the euclidean plane. Each one of these vertices is the centre of the circle "touching" a triple of objects (passing through points or tangent to any other geometric object). The algorithm starts with an initial triple of points pertaining to each one of the three objects. It computes its circumcentre and the closest point (called foot) of each object from the circumcentre. These three feet form the starting triple for the next iteration. We geometrically demonstrate a necessary and sufficient condition for the general case. This iterative algorithm is used as a new method for constructing a dynamic Voronoi diagram for a set of points and straight line segments (see Gold and *al.* [4]).

1 Introduction

The Voronoi diagram has many applications in a variety of disciplines, and has been widely treated in the literature (see Okabe and *al.* [5] and Aurenhammer [1] for a general survey). The Voronoi diagram has been introduced by the rusian mathematician Georgii Fedeorovitch Voronoi in a treatise on quadratic forms theory (see Voronoi [7], [8]). The ordinary point Voronoi diagram is a partition of the plane, in the way that each object (point) partitions the euclidean plane into a region, that is the locus of points which are closer from that object than from any other object (see Preparata and Shamos [6]). The concept of Voronoi diagram has been extended in different kinds of generalizations: higher order Voronoi diagrams (extension of the set S of generators, see Preparata and Shamos [6]), weighted Voronoi diagrams (see Okabe and *al.* [5]), Voronoi diagrams with obstacles (see Shamos and Hoey [1975] in Preparata and Shamos [6]), Voronoi diagrams for areas, and Voronoi diagrams for lines. The line Voronoi diagram is a generalization of the ordinary point Voronoi diagram, by extending the set S to points, line segments, and any "geometric element consisting of line segments that are connected" (see Okabe and *al.* [5]). The line Voronoi diagram has been intensively studied by Drysdale [1979], Lee [1978], Lee and Drysdale [1981] (in Okabe [5]), and Kirkpatrick [1979] in Okabe [5]. It is possible to distinguish different kinds of line Voronoi diagrams (see Okabe [5]): Voronoi diagram for a set of points and straight line segments, Voronoi diagram for a set of circles, and Voronoi diagrams for a set of points, straight line segments and circular arcs.

The Voronoi diagram for a set of geometric objects of the euclidean plane is defined by the generalization of the ordinary point Voronoi diagram by extending the set of objects S to any geometric element. This partition of the plane forms a net, whose vertices are called Voronoi vertices, and whose edges are called Voronoi edges. Each Voronoi vertex is the common intersection of exactly three edges, and therefore each Voronoi vertex is equidistant from its three nearest objects. An iterative algorithm has been used for "hunting Voronoi vertices in non polygonal domains" (see Ferruci and *al.* [3]). In their algorithm, the exact shape description of the objects is not needed. The only assumption is "to be able to answer to queries of the form "given a point p and an object S, determine the closest point on S from p"" (Ferruci and *al.* [3]). Starting from a point p on the plane, they compute the closest point on each object. Then, they compute the circumcentre of these three points, that will be the point p for the next iteration. They have defined a necessary condition of convergence, based on the fact that the smallest circle containing three points and whose centre is inside the triangle formed by these three points is the circle circumscribed to the three points. The sufficient condition is that the

next point p is inside the triangle formed by the closest point of each one of the three objects from the previous point p.

2 Preliminaries

Let $\mathbb{N}$ be the set of integers, $\mathbb{R}$ be the set of reals, and $\mathbb{R}^2$ be the euclidean plane. Let P be a point of $\mathbb{R}^2$, and O be a geometric object, then let's define the distance from P to O as: $d(P,O) = inf\{d(P,M)/M \in O\}$.

Let $\mathcal{O}$ be the set of the n generators of the Voronoi diagram.

$\mathcal{V}(\mathcal{O}) = \bigcup_{i=1}^{n} \mathcal{V}(O_i) = \mathbb{R}^2$ where $\mathcal{V}(O_i) = \{M \epsilon \mathbb{R}^2 / \forall j : d(M,O_i) \leq d(M,O_j)\}$ is the Voronoi region of the object O_i. Each Voronoi edge is a portion of bisector of two objects. These Voronoi edges intersect at points, called Voronoi vertices. Being the intersection of two bisectors, the Voronoi vertices are at the same distance from three objects.

Let H be the vectorial euclidean hyperplane corresponding to $\mathbb{R}^2$ in the oriented (see Berger [2]) three dimensional vectorial euclidean space E. Let $\overrightarrow{k}$ be the unitary vector of E normal to H.

Let O_1, O_2, and O_3 be three objects.

The iterative algorithm (see figure 1 page 3) starts with three arbitrary points (called feet) taken on each one of the three objects: F_{1_0}, F_{2_0}, and F_{3_0}. The centre C_0 of the circle $\mathcal{C}_0$ circumscribed to the triangle formed by these three feet is computed. Then, each one of the closest point of O_1, O_2, and O_3 from C_0: F_{1_1}, F_{2_1}, and F_{3_1} is computed and used as the starting point (foot) for the next iteration. The iterations stop when the distance between the present centre and the last one is smaller than a user-defined tolerance.

Let $(F_{1_n})_{n\in\mathbb{N}}$, $(F_{2_n})_{n\in\mathbb{N}}$, and $(F_{3_n})_{n\in\mathbb{N}}$ be the sequences of the points (called feet) on each one of the three objects O_1, O_2, and O_3, closest to the centre of $\mathcal{C}_{n-1}$ except for $n = 0$ where the foot are arbitrary points on each one of the objects.

Let $\mathcal{C}_n$ be the circle passing through F_{1_n}, F_{2_n}, and F_{3_n} for $n \geq 0$.

Let (C_n) be the sequence of the centres of the circles $\mathcal{C}_n$ for $n \geq 0$.

Let $\mathcal{C}_{1_n}$ be the circle whose diameter is $[F_{1_n}C_n]$ for $n \geq 0$.

Let $\mathcal{C}_{2_n}$ be the circle whose diameter is $[F_{2_n}C_n]$ for $n \geq 0$.

Let $\mathcal{C}_{3_n}$ be the circle whose diameter is $[F_{3_n}C_n]$ for $n \geq 0$.

3 A necessary and sufficient condition of convergence

First let's suppose that there exists a Voronoi vertex v for the triple of objects (O_1, O_2, O_3). Then, the circle whose centre is v and whose radius is the euclidean distance from v to O_1 touches the three objects O_1, O_2, O_3 respectively at P, Q, R in the counterclockwise order ($P \in O_1$, $Q \in O_2$, $R \in O_3$). This implies that the three feet are in the anticlockwise order (R is on the left of $\overrightarrow{PQ}$ or equivalently: $\overrightarrow{PQ} \times \overrightarrow{PR} \cdot \overrightarrow{k} \geq 0$) and $\overrightarrow{vQ}$ is between $\overrightarrow{vP}$ and $\overrightarrow{vR}$ (the oriented angles $\widehat{\overrightarrow{vP}\overrightarrow{vQ}}$ and $\widehat{\overrightarrow{vQ}\overrightarrow{vR}}$ are inferior to the oriented angle $\widehat{\overrightarrow{vP}\overrightarrow{vR}}$; see [2] for a survey of oriented angles).

Therefore, it is easy to see that the sequences of the feet $(F_{1_n})_{n\in\mathbb{N}}$, $(F_{2_n})_{n\in\mathbb{N}}$, and $(F_{3_n})_{n\in\mathbb{N}}$ should verify from some integer q, that the anticlockwise order of the feet is the expected one.

Now, let's suppose that we are at the iteration $n \geq q$ and the feet F_{1_n}, F_{2_n}, and F_{3_n}, are in the anticlockwise order.

We will consider now for each object O_i, the portion O_{i_n} of O_i inside the disk $\mathcal{D}_n$, whose boundary is $\mathcal{C}_n$. If O_i is a point, then $\forall i \in \mathbb{N} : O_{i_n} = O_i$. If $O_i \cap \mathcal{D}_n \neq \{F_{i_n}\}$ then we will consider O_{i_n} open, and otherwise we will consider O_{i_n} closed. If $\forall i \in \{1,2,3\} : O_i \cap \mathcal{D}_n = \{F_{i_n}\}$, then $\mathcal{C}_n$ is the circle touching the three Voronoi objects O_1, O_2, and O_3, at F_{1_n}, F_{2_n}, and F_{3_n} respectively. Its centre C_n is the Voronoi vertex corresponding to the triple of objects $\{O_1, O_2, O_3\}$. For each object O_i, any point of O_{i_n}, if it exists is closer from the centre of $\mathcal{C}_n$ than F_{i_n} and any other point of $O_i - O_{i_n}$.

Thus,

$$\forall i \in \{1,2,3\} : F_{i_{n+1}} \in O_{i_n}. \qquad (1)$$

If, and only if F_{1_n}, F_{2_n}, and F_{3_n} are in the anticlockwise order, $\overrightarrow{C_nF_{2_n}}$ is between $\overrightarrow{C_nF_{1_n}}$ and $\overrightarrow{C_nF_{3_n}}$ (the oriented angles $\widehat{\overrightarrow{C_nF_{1_n}}\overrightarrow{C_nF_{2_n}}}$ and $\widehat{\overrightarrow{C_nF_{2_n}}\overrightarrow{C_nF_{3_n}}}$ are inferior to the oriented angle $\widehat{\overrightarrow{C_nF_{1_n}}\overrightarrow{C_nF_{3_n}}}$; see figure 2 page 3). Indeed, $\overrightarrow{C_nF_{1_n}}$, $\overrightarrow{C_nF_{2_n}}$, and $\overrightarrow{C_nF_{3_n}}$ are three radiuses of the circle passing through the three feet F_{1_n}, F_{2_n}, and F_{3_n}. In the counterclockwise order along that circle, F_{1_n}, F_{2_n} and F_{3_n} are in same order as their radiuses from C_n: $\overrightarrow{C_nF_{1_n}}$, $\overrightarrow{C_nF_{2_n}}$, and $\overrightarrow{C_nF_{3_n}}$.

For each object O_i, $\mathcal{C}_n \cap \mathcal{C}_{i_n} = \{F_{i_n}\}$, and the common tangent of $\mathcal{C}_n$ and $\mathcal{C}_{i_n}$ is therefore the tangent of $\mathcal{C}_n$ at F_{i_n}. The edge orthogonal to the common tangent and passing through F_{i_n} is the edge

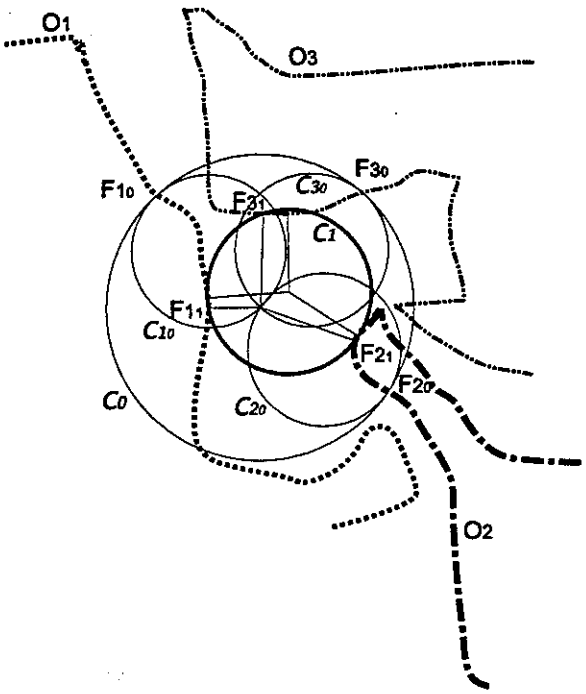


Figure 1: The iterative algorithm

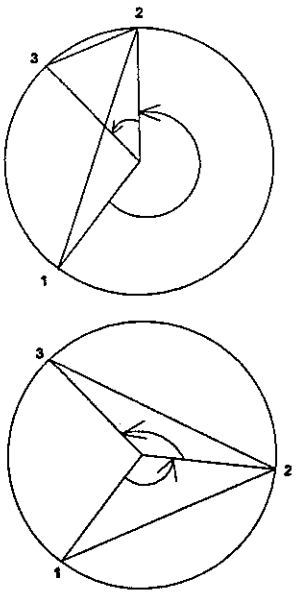


Figure 2: Visibility of three points from the circumcentre

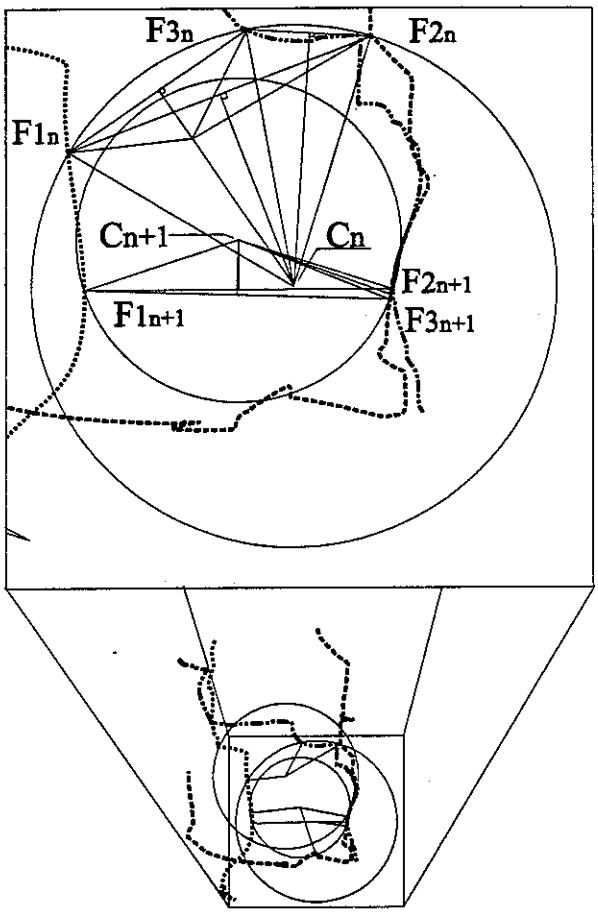


Figure 3: The convergence process when the order of the feet changes

$(F_{i_n}C_n)$. The vector $\overrightarrow{F_{i_n}C_n}$ gives the sense of the "movement" from F_{i_n} to $F_{i_{n+1}}$ along O_i, because it gives the relative position of O_{i_n} relatively to F_{i_n}.

As long as $\overrightarrow{F_{i_n}C_n} \cdot \overrightarrow{F_{i_{n+1}}C_{n+1}} \geq 0$, $F_{i_{n+2}}$ will be after F_{i_n} and $F_{i_{n+1}}$ along O_i. Now, let us see the case where at least for one object $\overrightarrow{F_{i_n}C_n} \cdot \overrightarrow{F_{i_{n+1}}C_{n+1}} < 0$. We will prove that if all the objects verify $\overrightarrow{F_{i_n}C_n} \cdot \overrightarrow{F_{i_{n+1}}C_{n+1}} < 0$, then the order of the feet $F_{1_{n+1}}$, $F_{2_{n+1}}$, and $F_{3_{n+1}}$ has changed relatively to the order of the feet F_{1_n}, F_{2_n}, and F_{3_n} (see figure 3 page 3).

$\overrightarrow{F_{i_n}C_n} \cdot \overrightarrow{F_{i_{n+1}}C_{n+1}} < 0$ is equivalent to $\overrightarrow{C_nF_{i_n}} \cdot \overrightarrow{C_{n+1}F_{i_{n+1}}} < 0$. Let m_{ij_n} be the middle of $[F_{i_n}F_{j_n}]$.

Then, $\widehat{\overrightarrow{C_nF_{i_n}}\overrightarrow{C_nm_{ij_n}}} = \widehat{\overrightarrow{C_nm_{ij_n}}\overrightarrow{C_nF_{j_n}}} = \frac{1}{2}\widehat{\overrightarrow{C_nF_{i_n}}\overrightarrow{C_nF_{j_n}}} = \widehat{\overrightarrow{F_{k_n}F_{i_n}}\overrightarrow{F_{k_n}F_{j_n}}}$ where O_k is the third object. From $\overrightarrow{C_nF_{i_n}} \cdot \overrightarrow{C_{n+1}F_{i_{n+1}}} < 0$ and $\overrightarrow{C_nF_{j_n}} \cdot \overrightarrow{C_{n+1}F_{j_{n+1}}} < 0$ we get $\overrightarrow{C_nm_{ij_n}} \cdot \overrightarrow{C_{n+1}m_{ij_{n+1}}} < 0$ by passing to the bisectors..

Because of the fact that the circle centre C_n passing through $F_{i_{n+1}}$ is tangent to O_i, either $[F_{i_n}F_{j_n}]$ and $\left[F_{i_{n+1}}F_{j_{n+1}}\right]$ are not hidden by any of the two other objects (if there is a valid circle touching the three objects in that order, no object is hiding completely O_i from the third object) or they are both hidden by another object. Therefore the relative positions of $[F_{i_n}F_{j_n}]$ and $\left[F_{i_{n+1}}F_{j_{n+1}}\right]$ are the same relatively to O_i and O_j, and we have:

$\overrightarrow{m_{ij_n}F_{j_n}} \cdot \overrightarrow{m_{ij_{n+1}}F_{j_{n+1}}} = \frac{1}{4}\overrightarrow{F_{i_n}F_{j_n}} \cdot \overrightarrow{F_{i_{n+1}}F_{j_{n+1}}} \geq 0.$

Therefore $\overrightarrow{C_nm_{ij_n}} \cdot \overrightarrow{C_{n+1}m_{ij_{n+1}}} < 0$ and $\overrightarrow{m_{ij_n}F_{j_n}} \cdot \overrightarrow{m_{ij_{n+1}}F_{j_{n+1}}} \geq 0$, and consequently $\left(\overrightarrow{C_nm_{ij_n}} \times \overrightarrow{C_nF_{j_n}}\right) \cdot \left(\overrightarrow{C_{n+1}m_{ij_{n+1}}} \times \overrightarrow{C_{n+1}F_{j_{n+1}}}\right) < 0$, and

$\left(\overrightarrow{F_{k_n}F_{i_n}} \times \overrightarrow{F_{k_n}F_{j_n}}\right)$

$\left(\overrightarrow{F_{k_{n+1}}F_{i_{n+1}}} \times \overrightarrow{F_{k_{n+1}}F_{j_{n+1}}}\right) < 0.$

That means that without any loose of generality, we passed from the anticlockwise order of F_{k_n}, F_{l_n}, F_{m_n} to the anticlockwise order of $F_{k_{n+1}}$, $F_{m_{n+1}}$, F_{l_n}. The objects whose foot order has been changed (l and m), either intersect between their two successive feet F_{l_n}, $F_{l_{n+1}}$, and F_{m_n}, $F_{m_{n+1}}$ respectively, or one of them (l or m) hides the other one from the third object (k). However, we supposed at the beginning, that the order of the feet was the final order in which the searched circle has to touch each object.

Thus, the fact that there is at least one object for which $\overrightarrow{F_{i_n}C_n} \cdot \overrightarrow{F_{i_{n+1}}C_{n+1}} > 0$, implies that $F_{i_{n+2}}$ can not be before F_{i_n} and $F_{i_{n+1}}$ along O_i, and therefore $F_{i_{n+2}}$ is either after F_{i_n} and $F_{i_{n+1}}$ or between F_{i_n} and $F_{i_{n+1}}$ along O_i. Therefore, if and only if, there exists a circle touching the three objects in that order, the sequences $(F_{i_n})_{n\in\mathbb{N}}$ will converge towards the closest points of O_i, $i \in \{1,2,3\}$ from the centre of that circle, and $(C_n)_{n\in\mathbb{N}}$ will converge towards the centre of that circle. We finally get the necessary and sufficient condition:

Theorem 1 *The necessary and sufficient condition of convergence of this iterative algorithm is the following one:*

$$\exists p \in \mathbb{N} / \forall n \geq p : \overrightarrow{F_{1_n}F_{2_n}} \cdot \overrightarrow{F_{1_n}F_{3_n}} \geq 0 \qquad (2)$$

From this necessary and sufficient condition of convergence of this iterative algorithm, we get directly the initial conditions for this algorithm:

Lemma 2 *If we start from three feet in the expected final order; and* $\forall (P,Q,R) \in O_{1_0} \times O_{2_0} \times O_{3_0} : \overrightarrow{PQ} \times \overrightarrow{PR} \cdot \overrightarrow{k} > 0$. *If, and only if, there exists a circle touching the three objects in the specified order, and for which there is no intersection with another object between the closest point of each object* O_i *from its centre and* F_{i_0}, *the sequences of* $(F_{i_n})_{n\in\mathbb{N}}$ *will converge towards the closest points of each one of the objects from the centre of that circle, and* $(C_n)_{n\in\mathbb{N}}$ *will converge towards the centre of that circle.*

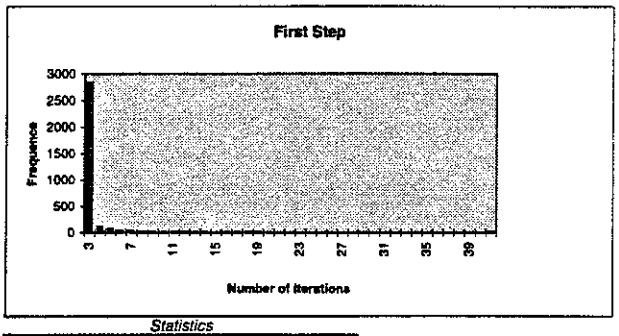
4 The Algorithm: description and statistical validity

An algorithm for the determination of Voronoi vertices for points and line segments has been developped using the precedent lemma. It is subdivided into three steps: two steps are necessary to satisfy the initial conditions, and the last step is the iterative algorithm itself.

In the first step, three starting feet in the expected order are chosen in order to satisfy to the first . Before trying to choose such feet, the extremities of the objects are checked to assess if it is possible. After, for each line segment object, a feet is randomly chosen till it is in the good order relatively to the other objects.

In the second step, the choice of the starting feet is corrected in order to satisfy the second initial condition: $\forall (P,Q,R) \in O_{1_0} \times O_{2_0} \times O_{3_0} : \overrightarrow{PQ} \times \overrightarrow{PR} \cdot \overrightarrow{k} > 0$.

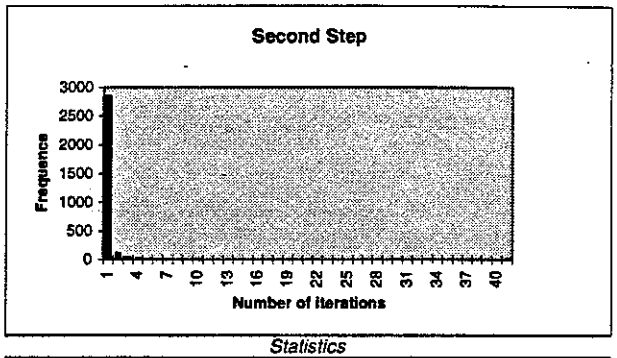
In order to do so, each previously chosen foot at the iteration n is replaced by a feet for which the two extremities of $O_{i_{n+1}}$ are in the good order relatively to the other objects. Implementation of



Statistics

Mean	4,820121951
Standard error	0,464252573
Standard deviation	26,58834109
Variance	706,9398817
Kurstosis	1207,749843
Asymmetry coefficient	32,70444668
Extent	1132
Minimum	3
Maximum	1135
Sum	15810
Number of observations	3280
Confidence value (95,0%)	0,910254764

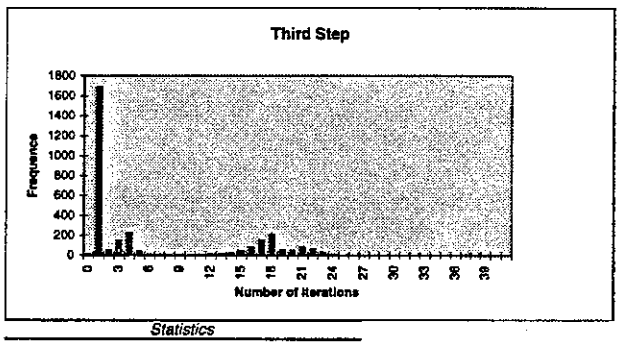
Figure 4:



Statistics

Mean	1,43818002
Standard error	0,104508027
Standard deviation	5,755537048
Variance	33,12620671
Kurstosis	292,9141068
Asymmetry coefficient	17,08998536
Extent	100
Minimum	1
Maximum	101
Sum	4362
Number of observations	3033
Confidence value (95,0%)	0,204913883

Figure 5:



Statistics

Mean	6,164164164
Standard error	0,138815491
Standard deviation	7,599434998
Variance	57,75141229
Kurstosis	-0,687893387
Asymmetry coefficient	1,04635822
Extent	27
Minimum	0
Maximum	27
Sum	18474
Number of observations	2997
Confidence value (95,0%)	0,27218343

Figure 6:

the above presented algorithm is done on points and line segments.

This algorithm has been implemented using Delphi on a Pentium based PC running Windows 95. Testing has been made using pseudo-random data including collisions. In order to generate special cases, we suppose that the second and third objects could be connected to the previous one.

The test data was composed of 10000 triples of objects. Among them, 3280 were assessed positively for circumcentre possibility. There was a valid circumcentre for 2997 cases, that is 91.37% of the previous set. The statistics for the 3280 cases mentioned above appear in the figures 4,5, and 6 page 5.

For the first step, we can see that 5 (4,82) iterations are needed in average. With a confidence interval of 95%, 6 (5,73) iterations are necessary.

For the second step, we can see that 2 (1,44) iterations are needed in average. With a confidence interval of 95%, 2 (1,64) iterations are necessary.

Finally, for the third step, we can see that 7 (6,16) iterations are needed in average. With a confidence interval of 95%, 7 (6,44) iterations are necessary.

5 Conclusions

This algorithm is presently being applied to the routine, that computes the centre of the circle that

touches three objects of a spatial data structure of points and oriented line segments. This routine is the fundamental part of a software of construction, and maintenance of a dynamic Voronoi spatial data structure for a set of points and oriented line segments. This spatial data structure is currently developed at the Industrial Chair of Geomatics applied to Forestry of the Centre for Research in Geomatics of Laval University, Quebec City (Canada) by Dr Christopher M. Gold.

Finally, this iterative algorithm is particularly interesting because the mathematical calculations involved in it (closest point and determinants) are directly transposable to the sphere.

6 Acknowledgments

This research work has received the financial support of the Natural Sciences and Engineering Research Council (NSERC) of Canada, and the Association des Industries Forestières du Québec (A.I.F.Q.). Finally, I am grateful to Dr. Jack Snoeyink (UBC), and Ms. Darka Mioc, Ph. D. candidate at Laval University, for their comments and suggestions.

References

[1] F. Aurenhammer, Voronoi diagrams - A survey, (Institute for Information Processing, Technical University of Graz, Report 263)

[2] M. Berger, Géométrie, volume 2 : espaces euclidiens, triangles, cercles et sphères, (CEDIC/FERNAND NATHAN, Paris, 1979) 216p.

[3] V. Ferruci, M. Overmars, A. Rao, and J. Vleugels, Hunting Voronoi Vertices in Non-Polygonal Domains, (CCCG'94, Saskatoon, Canada, 1994) 45-50

[4] C.M. Gold, P.R. Remmele, and T. Roos, Voronoi Diagrams of Line Segments Made Easy, (CCCG'95, Québec, Canada, 1995) 223-228

[5] A. Okabe, B. Boots, and K. Sugihara, Spatial Tessellations, Concept and Applications of Voronoi Diagrams, (John Wiley & Sons, 1992) 532 p.

[6] F.P. Preparata, and M.I. Shamos, Computational Geometry, An Introduction, (Springer-Verlag, 1985) 398 p.

[7] G. Voronoi, Nouvelles applications des paramètres continus à la théorie des formes quadratiques, Premier Mémoire, Recherches sur les paralléllòèdres primitifs, (Journal fur die Reine und Angewandte Mathematik, v. 134, 1908) 198-287

[8] G. Voronoi, Nouvelles applications des paramètres continus à la théorie des formes quadratiques, Deuxième Mémoire, Recherches sur les paralléllòèdres primitifs, (Journal fur die Reine und Angewandte Mathematik, v. 136, 1909) 67-181

Some Tools for Modeling and Analysis of Surfaces

Carsten Dorgerloh* *University of Bonn*
Jens Lüssem† *University of Bonn*
Morakot Pilouk‡ *ESRI-Redlands*
Jürgen Wirtgen§ *University of Bonn*

April 14, 1997

Abstract

We present some algorithms which construct a triangular graph given a set of points, where each face of the graph complies with the Delaunay criteria. Then we develop an $\mathcal{O}(n)$ algorithm to construct the contour of such a triangular graph, where each face is coloured either black or white. Our techniques avoid expensive trigonometric computations. We intoduce our algorithms for the 2-dimensional case and show how to extend them to the d-dimensional case in a straightforward manner.

1 Introduction

Consider the following scenario: An oil company drills n boreholes $p_1, \ldots, p_n$. With each p_i an value $val(p_i)$ will be associated, which represents the expected amount of oil supposed to be in the neighborhood of p_i. They want to determine which connected regions in the plane promise to give a lot of oil.
In order to solve problems of this type, we perform three phases.

Triangulation phase: Based on the n points, we construct a planar graph structure T consisting of non-intersecting triangles.

Rating phase: For each triangle $t = (p_1, p_2, p_3)$ (with p_1, p_2, p_3 being the vertices of t) we evaluate $f(p_1, p_2, p_3)$, where f is an appropriate rating, e.g. the mean of the $val(p_i)$. Depending on some threshold we color the triangle black or white.

Contour phase: In this phase we compute the set of cycles C defined by edges of T, which are on a common boundary of a black and a white triangle. Furthermore, C is constrained to contain only those cycles which are not enclosed by another cycle.

With growing n, this construction tends to be similar to the real world, since more points give us more information. Therefore our modeling of the reality will become more and more close grained.
The paper is organized as follows. Section 2 describes the triangulation phase. To be more precise, we discuss an triangulation algorithm which complies with the Delaunay criteria. Section 3 contains the elementary steps of the algorithm for the contour problem and shows how to implement them efficiently. From the proof of correctness of the elementary steps the correctness of the main algorithm of the sequence is immediately clear. Finally, we describe that the algorithm can easily be extended to the d-dimensional case.

2 Constructing the Delaunay triangulation

Triangulation has been applied in many disciplines especially for modeling and analysis of surface, e.g. terrain modeling in GIS, civil engineering, landscape architecture. From a triangulated structure representing a surface, we can compute slope, aspect, visibility, isolines, light reflectance from the surface, volume above or below the surface with respect to a given datum. For our algorithms, we use the Delaunay triangulation, having some nice properties [Au 91].

Definition 1 *Let $\mathcal{P} = \{p_1, ..., p_n\} \subseteq \mathbb{R}^d$ be a set of d-dimensional points. For all $p \in \mathcal{P}$ we define the Voronoi cell $V(p)$ to be the subset of $\mathbb{R}^d$ which is closer to p than to any other point in $\mathcal{P}$. Formally*

$$V(p) = \{x \in \mathbb{R}^d : \|x - p\| < \|x - q\| \, \forall q \in \mathcal{P} \setminus \{p\}\}$$

*Institut für Informatik, Univ. Bonn, Römerstr. 164, 53117 Bonn, Germany, email: carsten@cs.uni-bonn.de
†Institut für Informatik, Univ. Bonn, Römerstr. 164, 53117 Bonn, Germany, email: jens@cs.uni-bonn.de
‡ESRI, 380 New York Street, Redlands, CA 92373, USA, email: mplouk@esri.com
§Institut für Informatik, Univ. Bonn, Römerstr. 164, 53117 Bonn, Germany, email: wirtgen@cs.uni-bonn.de

Let $H(p,q)$ the halfspace defined through the bisecting hyperplane of p and q containing p. So we have another definition of $V(p)$:

$$V(p) = \bigcap_{q \in \mathcal{P} \setminus \{p\}} H(p,q)$$

The cells $V(p)$ partition the $\mathbb{R}^d$ and form by this way the Voronoi diagram. The dual graph will be called the Delaunay triangulation. Here we have some elementary properties of the Delaunay triangulation.

Property 1

1. *The delaunay triangulation is a partition of the $\mathbb{R}^d$ in simplices, whose vertices are the points of $\mathcal{P}$.*

2. *The Delaunay triangulation of a Voronoi diagram is unique:*

   The cells which circumscribe the simplices do not contain any point of $\mathcal{P}$ in their interior.

This property gives us the simple algorithm *SimpleDelaunay*($\mathcal{P}$) with a worst-case complexity of $O(n^{\lceil n/2 \rceil + 1})$ [Bo 81].

SimpleDelaunay($\mathcal{P}$)
Input: A set $\mathcal{P} = \{p_1, ..., p_n\} \subseteq \mathbb{R}^d$ of points.
Output: The Delaunay triangulation of $\mathcal{P}$.

1. Let $s_1 := (p_1, ..., p_{d+1})$ the first simplex.

2. Define $S = \{s_1\}$ to be the set of actual simplices. Set *card_simp* := 1, the number of actual simplices.

3. For all $i = d+2, ..., n$ do

   (a) $R(p_i) := \emptyset$. In the following $R(p_i)$ will be the region defined by the union of the simplices in S, whose circumscribing ball contain p_i.

   (b) For all $j = 1, ..., card\_simp$ do

   - If *circumball*(p_i, s_j) then
     - $R(p_i) := R(p_i) \cup s_j$,
     - $S := S \setminus \{s_j\}$,

   (c) Let $F(p_i)$ the set of facets defined by $R(p_i)$.

   (d) Construct new simplices by connecting p_i to the elements of $F(p_i)$.

   (e) Update the number *card_simp*

To complete this algorithm we have to describe the subroutine *circumball*(p,s) which returns *true* iff p is contained in the circumscribing ball of the simplex s. The easiest way to do this, is to compute this ball and test whether p is inside or not.
In the 2-dimensional case, we know from the elemental geometry, that the center of the circumscribing circle of a triangle is defined by the intersecting point of the orthogonals on the centers of the sides. It suffices to calculate two of these and solving the linear system.
It is easy to see, that we can generalize it to d dimensions with the following recursive procedure *ConstructCircumball*(s,d):

ConstructCircumball$_d(s)$
Input: A simplex $s = (p_1, ..., p_{d+1})$.
Output: The center of the circumscribing ball of s.

1. If $d = 1$, return the center of s. (s is a line segment)

   Else, take two arbitrary facets f_1 and f_2 of s and calculate

   - $c_1 = ConstructCircumball_{d-1}(f_1)$,
   - $c_2 = ConstructCircumball_{d-1}(f_2)$.

2. Construct the orthogonals o_1 and o_2 on f_1 and f_2 thru c_1 and c_2.

3. Calculate the intersection of o_1 and o_2, which gives us the center c.

4. Return c.

Now it is easy to implement *circumball*(p,s). We calculate the center c of the ball which circumscribes the simplex s and take some arbitrary vertex p_i $i = 1, ..., d+1$ of s. Now we have only to check whether

$$\|c - p_i\| \geq \|c - p\|,$$

or not.

3 The contour algorithm

Before we explain the elementary steps of the contour algorithm we need the following definitions.
Let $G = (V, E)$ be a planar graph. Consider a fixed plane embedding of G. The unbounded region is called the *exterior face*. Other faces are called *interior faces*. The vertices and the edges on the exterior face are called *exterior vertices* and *exterior edges*, respectively. For each vertex v, $N(v)$ denotes the set of

neighbors of v. A planar graph $T = (V_T, E_T)$ has a *triangular embedding*, if every face of T, except the exterior face, is a triangle. The triangles of a *colored triangular embedding* are colored black and white, respectively. The color of the external face is always assumed to be white. Let S be a set that contains all edges of T which are on a common boundary of a black and a white triangle. In fact, S is a collection of edge-disjoint simple cycles (the points of the cycles being evident by context) of T, which are separating white and black regions. The problem of finding simple cycles is one of the most basic and natural algorithmic graph problems (see [Le 90]) and was considered by many researchers e.g. [Mo 85], [AYZ 95], [DW 97a], [DW 97b]. However, in the present paper we search for a special set of cycles - namely the external contour.

An *(external) contour* C (introduced in [DL 95]) of a colored triangular embedding T consists of those cycles of S, which are not enclosed by another cycle of S. The task of computing the external contour of T can now be formalized to produce the set C.

The following well known lemma (see e.g. [Ha 69] [Ev 79]) is helpful, because for planar graphs it implies that an $\mathcal{O}(|V|+|E|)$ algorithm is really an $\mathcal{O}(|V|)$ algorithm.

Lemma 2 *If $G = (V, E)$ is any planar graph with $|V| \geq 3$. Then G has at most $3|V| - 6$ edges.*

The algorithm is built of the following steps.

3.1 Construction of the dual graph

Given a colored triangular embedding T, its *colored dual* $G = T^*$ is constructed as follows: simply trace the boundary of each face, place a vertex in G for each face of T (excluding the exterior face) and assign the corresponding color to it. If two faces of T have an edge e_T in common, join the corresponding vertices in G by an edge e.

It is quite straightforward to solve the problem in $\mathcal{O}(|V|)$ time sequentially, if we trace the boundaries by following cyclic linked lists.

In the following we denote by *construct_dual*(T) the procedure that executes the step as described above.

3.2 The extension of the dual

Now, we extend G by introducing a new vertex v_{ext}. v_{ext} corresponds to the external face of T and is assigned the color white. We connect this vertex to each node $u \in V - \{v_{ext}\}$ with $degree(u) < 3$ (see Figure 1). That are exactly those vertices in G corresponding to faces in T, which have a common boundary with the external face of T. Hence, adding v_{ext} cannot destroy the planarity of G.

```
extend(G)
   V := V ∪ {v_ext}
   for each u ∈ V − {v_ext} do
      if degree(u) < 3 then
         E := E ∪ {u, v_ext}
      end if
   end for
```

Figure 1: Extend G by v_{ext}

```
hollow_out(G)
   S := N(v_ext)
   while S ≠ ∅ do
      u := first element of S
      if colour(u) = white then
         for each v ∈ N(u) − {v_ext} do
            if v_ext ∉ N(v) then
               E := E ∪ {v, v_ext}
            E := E − {v, u}
            S := S ∪ {v}
         end for
         E := E − {u, v_ext}
         V := V − {u}
      else
         mark(u)
      end if
      S := S − {u}
   end while
```

Figure 2: Formulation of the procedure *hollow_out*

3.3 The hollow out step

What is the purpose of introducing the special vertex v_{ext}? The reason is, that we are now able to hollow out the white "regions" of T starting at the external face of T. This is done by changing G, while T remains unchanged. Furthermore, black vertices corresponding to faces in T which contribute edges to the contour of T are marked. The procedure which executes this step is given in Figure 2.

Before we prove the correctness of *hollow_out*(G), we

```
remove_v_ext(G)
    for all u ∈ N(v_ext)
        E := E − {u, v_ext}
    end for
    V := V − {v_ext}
```

Figure 3: Removal of v_{ext}

need the following definition. We say a vertex $u \in V$ is *enclosed by black*, if there is no path from v_{ext} to u in G such that all vertices on that path, except maybe u, are coloured white.

Lemma 3 *hollow_out(G) applied to the dual $G = (V, E)$ (extended by v_{ext}) of any triangular embedding of $T = (V_T, E_T)$ requires $\mathcal{O}(|V|)$ time.*

PROOF: S initially contains all neighbors of v_{ext}. In the course of the algorithm the set S is modified as follows. In each execution of the while-loop one vertex $u \in S$ is picked. If the color of u is white, then all neighbors of u, except v_{ext}, are inserted into S. Furthermore, G is changed: each $v \in N(u) - \{v_{ext}\}$ is connected to v_{ext}, u and all incident edges are deleted from G. On the other hand, if the color of u is black, then u is marked. Finally, u is deleted from S. It follows by an inductive argument, that every white vertex which is not enclosed by black is deleted from G. Moreover, every black vertex which is not enclosed by black is marked.
Lemma 2 guarantees that the while loop is executed at most $\mathcal{O}(|V|)$ times. Each of the $\mathcal{O}(|V|)$ runs of the while loop needs constant time because each vertex $u \in V - \{v_{ext}\}$ has at most three neighbors. ■

3.4 Finding the black components

In this step we first remove the vertex v_{ext} and all edges which are incident to that node from G (see Figure 3). The remaining graph is made up of what we call the ***black components*** of G. The computation of the black components can be done using standard algorithms for connected components which are based on depth-first-search or breadth-first search (see e.g. [AHU 83]). This step runs in $\mathcal{O}(max(|V|, |E|))$ time and is denoted by *compute_black_components(G)*.

3.5 Computation of the external contour

We describe the procedure *contour_of_component(G_i, T, G, G_{init})*, which computes the external contour of a component G_i of G. The structure of the procedure will be immediately clear from the following lemma. Again, we need several definitions. Each $u \in V$ corresponds to a face in T. Let us denote by $\lambda(u)$ the set of vertices and by $\gamma(u)$ the set of edges of the corresponding triangle in T. By G_{init} we denote the graph produced by *construct_dual(T)*.

Lemma 4 *The edge list returned by contour_of_component(G_i, T, G, G_{init}) is the external contour of G_i. contour_of_component(G_i, T, G, G_{init}) runs in time $\mathcal{O}(|V|)$.*

PROOF: Since only the faces corresponding to marked vertices contribute edges to the external contour $E_{contour}(i)$ of G_i, it suffices to investigate only such vertices. Let u be a marked vertex with $degree_{G_{init}}(u) = 3$ and $v \in N_{G_{init}}(u)$. Let $e_c \in E_T$ be the common edge of the faces corresponding to the vertices u and v, respectively. Furthermore, assume that $colour(v) = white$. We claim that $e_c \in E_{contour}(i)$. Suppose, to the contrary, that $e_c \notin E_{contour}(i)$. As an immediate consequence, the other edges of $\gamma(u)$ are not in $E_{contour}(i)$. But this implies that u is not marked, which is a contradiction. The other cases consider exterior edges and can be proven similarly. Since the computation of a common edge of two faces takes $\mathcal{O}(1)$ time, *contour_of_component(G_i, T, G, G_{init})* runs in time $\mathcal{O}(|V|)$. ■

3.6 The main procedure

Now we present the main procedure, *external_contour* (see Figure 4), putting the developed things together. We summarize the analysis of the previous section in the following theorem.

Theorem 5 *The external contour of a triangular graph can be computed in time $\mathcal{O}(|V|)$.*

An algorithm with this complexity is given explicitely.

3.7 Extension to the d-dimensional case

The above algorithm generalizes to the d-dimensional ($d > 2$) case where we consider d-simplexes.

```
external_contour()
    construct_dual(T)
    extend(G)
    hollow_out(G)
    remove_v_ext(G)
    compute_black_components(G)
    for each black component G_i of G
        contour_of_component(G_i, T, G, G_init)
```

Figure 4: The main procedure

First, we illustrate the changes neccessary by describing the 3-dimensional case. Here, we have to consider tetrahedrons instead of triangles. The dual graph is now constructed by identifing each tetrahedron by a vertex. Two vertices are joined by an edge, if their corresponding tetrahedrons have a face in common. Thus, each vertex in the dual graph has at most four neighbours. The adaption of the other steps is straightforward and it can easily be shown that the algorithm runs in time linear in the number of tetrahedrons.
This method can be extended to the d-dimensional ($d > 3$) case as well: Two d-simplexes are adjacent if they have a $(d-1)$-simplex in common. A vertex in the dual graph has at most $d+1$ neighbours. Again, it can be shown that the algorithm runs in time linear in the number of d-simplexes.

References

[AHU 83] Aho, A., Hopcroft, J., Ullman, J., *Data Structures and Algorithms*, Addison-Wesley Publishing Company, 1983.

[AYZ 95] Alon, N., Yuster, R., Zwick, U., *Color-coding*, Proc. 42^{nd} Journal of the ACM (1995), pp. 844–856.

[Au 91] Aurenhammer, F., *Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure*, ACM Computing Surveys **23**(3) (1991), pp. 345–405.

[Bo 81] Bowyer, A., *Computing Dirichlet tessellations*, Computer J. **24** (1981), pp. 162–166.

[DL 95] Dorgerloh, C. F., Lüssem, J., *A simple linear-time algorithm to find the contour in a coloured triangular graph*, Research Report 85146-CS, Institut für Informatik der Universität Bonn, 1995.

[DW 97b] Dorgerloh, C. F., Wirtgen, J., *Once again: Finding simple cycles*, Research Report 85165-CS, Institut für Informatik der Universität Bonn, 1997.

[DW 97a] Dorgerloh, C., Wirtgen, J., *Faster Finding of Simple Cycles in Planar Graphs on a randomized EREW-PRAM*, Proc. 2^{nd} Workshop on Randomized Parallel Computing (1997), held in conjunction with IPPS'97.

[Ev 79] Even, S., *Graph Algorithms*, Computer Science Press, 1979.

[Ha 69] Harary, F., *Graph Theory*, Addison-Wesley Publishing Company, 1969.

[Le 90] Leeuwen, J. v., *Graph Algorithms. Handbook of Theoretical Computer Science, Volume A, Algorithms and Comlexity*, chapter 10, pp. 525–631, Elsevier and The MIT Press, 1990.

[Mo 85] Monien, B., *How to find long paths efficiently*, Annals of Discrete Mathematics **25** (1985), pp. 239–254.

An increasing-circle sweep-algorithm to construct the Delaunay diagram in the plane

- Extended abstract -

B. Adam, P. Kauffmann, D. Schmitt and J.-C. Spehner
Laboratoire MAGE, Université de Haute-Alsace, 68093 Mulhouse, France
[B.Adam|P.Kauffmann|D.Schmitt|spehner]@univ-mulhouse.fr

Abstract

We present a new way to compute the Delaunay diagram of a planar set S of n sites in $O(n \log n)$ time by using a plane sweep technique. We sweep the plane by a circle whose center is a fixed point in the convex hull of S and whose radius increases from 0 to $+\infty$. This method is interesting notably when the diagram has to be constructed locally around a given point. We do not know of any method to reduce the sweep circle algorithm to a sweep line algorithm.

Key-words : Planar site Delaunay diagram, plane sweep algorithm.

1. Introduction

The Delaunay diagram, Delaunay triangulation and Voronoi diagram are well known structures in computational geometry and are used in various domains such as crystallography, physic, CAM-CAD, archaeology, ... [3, 5, ...]. One of the major method to construct these diagrams consists in sweeping the plane by a line. When the line sweeps over a new site, the current Voronoi diagram is updated by finding the region of this diagram in which the site is located [7]. Dually, it comes down to finding the site of the current Delaunay diagram to which the swept site must be connected [8].

In this paper, we introduce a new algorithm that computes the Delaunay diagram by sweeping the plane with an increasing circle.

This algorithm updates the current Delaunay diagram by processing two kinds of events : the «site event» and the «ultimate point event».

We first define the location structures that will allow us to connect a newly swept site to the already validated Delaunay diagram. Then we show how the algorithm detects and processes the events.

Finally, we show that our algorithm constructs the Delaunay diagram in $O(n \log n)$ worst case running time and we compare it with the sweep line algorithm.

2. The definition of Del(S)

Let E be the Euclidean plane and $d(x,y)$ the Euclidean distance between two points x and y of E. Let S be a planar set of n points of E called *sites*.

Every circle σ of E that contains no site in its interior is said to be *S-critical* and the set of sites which are on σ is called a *section*.

The Delaunay diagram of S is a partition of E whose vertices are the sites of S. For every section $\{s,t\}$, the open straight line segment

e(s,t) linking s to t is an *edge* of Del(S). For every section Q of dimension 2, the interior r(Q) of the convex hull of Q is a *region* of Del(S). If $|Q|>3$, r(Q) is not a triangle and Del(S) is not a triangulation.

The complementary of the convex hull of S is the only unbounded region of Del(S).

3. The validated Delaunay diagram Del(T,ρ) and its front $\mathcal{F}$

Let a point O in the convex hull of S be the origin of the polar coordinates and the center of the sweep circle C. S is sorted using the sites' radiuses. Two sites having the same radius are not ordered. If T is the set of swept sites, then $\forall\, t \in T$, $\forall\, s \in S\backslash T$, $\rho_t = d(O,t) < d(O,s) = \rho_s$ and the radius of the sweep circle C is $\rho = \max\{d(O,t);\, t \in T\}$. Every T-critical circle contained in C is also S-critical.

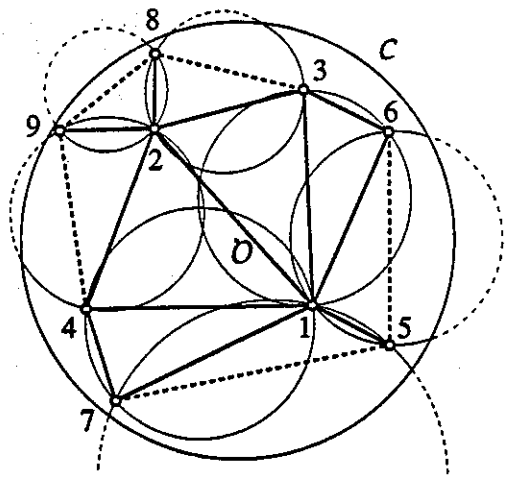


Figure 1. The validated Delaunay edges in full lines and the non-validated edges in dashed lines. The sites are numbered according to their distance to O. Front $\mathcal{F}$ is (1, 5, 1, 7, 4, 2, 9, 2, 8, 2, 3, 6, 1).

An *edge* e(s,t) [resp. a *region* r(Q)] of Del(T) is said to be *validated* if there exists a T-critical circle contained in C that goes through s and t [resp. all sites of Q]. The sub-diagram Del(T,ρ) of Del(T) built with the sites of T and the validated edges and regions of Del(T) is called *the validated Delaunay diagram of* (T,ρ).

For every edge e(s,t) of Del(T,ρ), st denotes the oriented straight line segment of origin s and end point t. Every edge and every vertex of the unbounded region of Del(T,ρ) is said to be *frontal*. If pq and qr are two consecutive edges of this unbounded region, the *triple* (p,q,r) is said to be *frontal*. The sequence $\mathcal{F}$ of all consecutive frontal edges is called *the Delaunay diagram front*.

4. The location of a point of C in $\mathcal{F}$

When C increases and sweeps over a site s, we search a site t in $\mathcal{F}$ such that s and t can be linked together to form a validated Delaunay edge. Therefore we define a partition of C which is dual to $\mathcal{F}$ and we show how to locate a site s in this partition.

4.1 Definitions of the point and the arc attached to a frontal edge

If pq is a frontal edge, there exists a unique circle C(p,q) going through p and q that is tangent to C at a point on the left of pq and such that C(p,q) is contained in C. This contact point is called the *point attached* to the edge pq and is denoted by pat(p,q).

Lemma 1. The circle C(p,q) is S-critical. Hence, if pat(p,q) is a site, pqpat(p,q) is a validated Delaunay region.

Let (p,q,r) be a frontal triple, the set of points of C that are on the left of pat(p,q)pat(q,r) is called the *arc attached* to the triple (p,q,r) and is denoted by arcat(p,q,r). arcat(p,q,r) is an open circular arc.

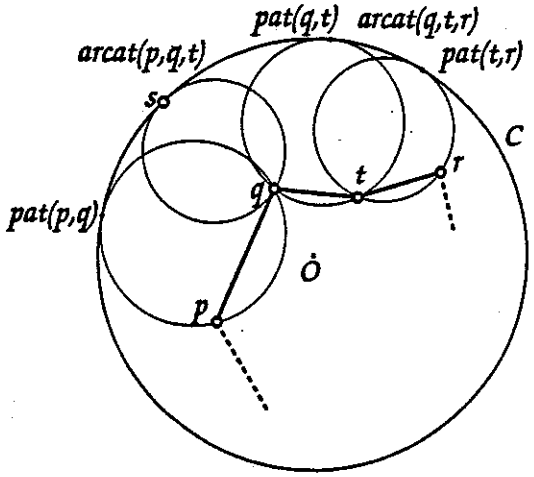


Figure 2. The frontal partition of C dual to $\mathcal{F}$.

Lemma 2. For every point s of arcat(p,q,r), the circle D(q,s) tangent to C at s that goes through q is S-critical. Hence, if $s \in S$, the open straight line segment qs is a validated Delaunay edge.

According to these two lemmas each newly swept site is linked to the validated Delaunay diagram. This proves that Del(T,ρ) is connected.

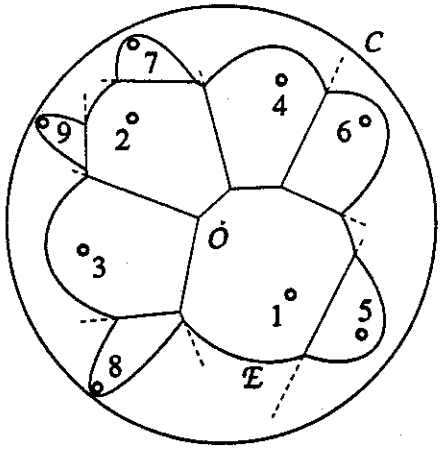


Figure 3. The elliptic front and the definitively constructed restriction of Vor(S) (full lines).

Remark. For every arcat(p,q,r) the set of the centers of the circles D(q,s) is an elliptic arc. The restriction of Vor(S) that is inside the union $\mathcal{E}$ of all the elliptic arcs is definitively constructed. $\mathcal{E}$ corresponds to the parabolic front of the sweep line algorithms [4, 8, 9].

4.2 The frontal partition of C and the location test

The partition of C in vertices, that are the attached points, and in arcs, that are the attached arcs, is called the *frontal partition of* C. The mapping ψ from $\mathcal{F}$ to the frontal partition of C such that, for every frontal edge pq, $\psi(pq)$=pat(p,q) and, for every frontal triple (p,q,r), $\psi(p,q,r)$=arcat(p,q,r) is a one-to-one duality by lemma 1 and 2.

To locate a point s in the frontal partition of C we use a binary balanced search tree BT($\mathcal{F}$). To each frontal edge is associated a node of BT($\mathcal{F}$). A frontal edge is inserted in BT($\mathcal{F}$) when created and is removed when it is no longer frontal. The frontal edges are sorted accordingly to their order in $\mathcal{F}$. Thus we can locate a point s of C in logarithmic time in the number of frontal edges. The location test is used in section 6 to create Delaunay edges.

5. Collision of attached points and determination of Delaunay regions

If (p,q,r) is a frontal triple such that r is on the left of pq, the circumcircle of the triangle pqr is denoted by circ(p,q,r). The point of circ(p,q,r) which is farthest from O is called the *ultimate point* associated to the frontal triple (p,q,r) and is denoted by ult(p,q,r). ult(p,q,r) is the last point of circ(p,q,r) to be swept and stays *alive* until it is *killed* when (p,q,r) is no longer frontal. When the radius of C increases, pat(p,q) and pat(q,r) tend towards each other. If circ(p,q,r) is S-critical, they collide in ult(p,q,r); in this case ult(p,q,r) is said to be *validated* and circ(p,q,r) circumscribes a validated Delaunay region.

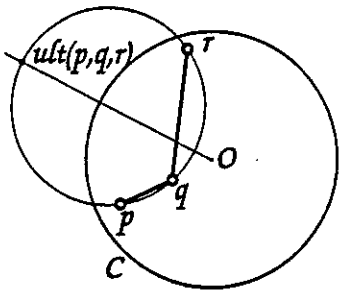


Figure 4. The ultimate point associated to (p,q,r).

If $k \geq 4$ consecutive frontal sites s_i, s_{i+1}, ..., s_{i+k-1} are cocircular, k-2 triples of $\mathcal{F}$ have overlapping ultimate points. This simple test detects all cocircular sites and is used to build the non triangular regions.

To be able to find the ultimate point closest to O, the ultimate points are inserted in a binary balanced search tree BT($\mathcal{U}$). An ultimate point is inserted in BT($\mathcal{U}$) when created and is deleted from BT($\mathcal{U}$) when killed or validated.

6. Processing the algorithm events

An event is a moment in the algorithm when C sweeps over a site or an ultimate point. The events add Delaunay edges and regions to Del(T,ρ). Thus we need to update the trees BT($\mathcal{F}$) and BT($\mathcal{U}$).

6.1 The event "site which is not an ultimate point"

6.1.1 Updating BT($\mathcal{F}$).

If $s \in$ arcat(p,q,r), the Delaunay edge e(s,q) is created in BT($\mathcal{F}$). Otherwise s overlaps the point attached to a frontal edge pq. In this case the edge pq is unique and the Delaunay edges e(s,p) and e(s,q) are created in BT($\mathcal{F}$). The former case implies the creation of a Delaunay region.

6.1.2 Updating BT($\mathcal{U}$).

If $s \in$ arcat(p,q,r), the ultimate point associated to the triple (p,q,r) is killed, if it exists. Otherwise s overlaps pat(p,q). If p' and q' are respectively the sites preceding p and following q in $\mathcal{F}$; the ultimate points associated to the triples (p',p,q) and (p,q,q') are killed, if they exist.

Moreover, if $s \in$ arcat(p,q,r) [resp. s = pat(p,q)], the new frontal triples (p,q,s) and (s,q,r) [resp. (p',p,s) and (s,q,q')] may generate new ultimate points.

6.2 The event "ultimate point which is not a site"

When sweeping an ultimate point u, there exists a sub-path $(p_1,p_2,\ldots,p_k)$ of $\mathcal{F}$ with $k \geq 3$ in which all the frontal triples have ultimate points overlapping u.

6.2.1 Updating BT($\mathcal{F}$).

We create the Delaunay edge e(p_1,p_k) in BT($\mathcal{F}$) which closes the Delaunay region of vertices $p_1, p_2, \ldots, p_k$. We delete from BT($\mathcal{F}$) the edges p_ip_{i+1}, $\forall\, i \in [1,k\text{-}1]$.

6.2.2 Updating BT($\mathcal{U}$).

The ultimate points associated to the triples (p_i,p_{i+1},p_{i+2}), $\forall\, i \in [1,k\text{-}2]$, are killed. The new frontal triples (p_0,p_1,p_k) and (p_1,p_k,p_{k+1}) may generate new ultimate points.

6.3 The event "ultimate site"

This event is a mix of the two previous ones and occurs when a site s overlaps ultimate points ult(p_i,p_{i+1},p_{i+2}), $\forall\, i \in [1,k\text{-}2]$.

6.3.1 Updating BT($\mathcal{F}$).

We create the Delaunay edges e(p_1,s) and e(s,p_k) in BT($\mathcal{F}$) that close the Delaunay region

of vertices $s, p_1, p_2, \ldots, p_k$. Thus we delete from $BT(\mathcal{F})$ the edges p_ip_{i+1}, $\forall\, i \in [1,k\text{-}1]$, that are no longer frontal.

6.3.2 Updating $BT(\mathcal{U})$.

We kill the ultimate points associated to (p_i,p_{i+1},p_{i+2}), $\forall\ i \in [1,k\text{-}2]$. The new frontal triples (p_0,p_1,s) and (s,p_k,p_{k+1}) may generate new ultimate points.

6.4 The algorithm main lines

The sites are stored in a sorted list $Lst(\mathcal{S})$ following the increasing radius order. We use a map [6] to represent $Del(T,\rho)$. The algorithm ends when C has swept all the sites ($Lst(\mathcal{S}) = \emptyset$) and when all the Delaunay regions are constructed ($BT(\mathcal{U}) = \emptyset$).

```
Delaunay_diagram (Lst(S):Sorted sites) → M:Map {
  BT(U) ← ∅; BT(F) ← ∅; M ← ∅;
  while Lst(S) ∪ BT(U) ≠ ∅ {
    extract the minimum e of Lst(S) ∪ BT(U);
    if e is a site which is not an ultimate point then {
      locate e in BT(F);
      if e = pat(p,q) then
        create the edges ep and eq in M;
      otherwise create the edge eq in M; }
    otherwise if e is an ultimate site then
      create the edges ep1 and epk in M;
    otherwise
      create the edge p1pk in M;
    update BT(F);
    update BT(U); }
  return M; }
```

7. The algorithm complexity

Lemma 3. The number of frontal edges of $\mathcal{F}$ and the number of ultimate points alive are less than or equal to $2|T| - 2$.

Proof. From 6, when a site is swept, except for the first one, at most two frontal edges are created. When an ultimate point which is not a site is swept, at least one frontal edge is removed. Thus the number of frontal edges is less than or equal to $2|T| - 2$. The number of ultimate points alive is also less than or equal to $2|T| - 2$ since every ultimate point is associated to a frontal triple. □

Theorem. If S is a planar set of n sites, the increasing circle algorithm computes $Del(S)$ in $O(n \log n)$ time.

Proof. Since $Del(S)$ is a partition of the plane with n vertices, $Del(S)$ admits at most $2n$-4 regions. Using 6.1, 6.2 and 6.3, for every ultimate point swept, a face is validated and, for every site swept, a vertex is validated. Thus the algorithm handles at most $3n$-4 events. Using lemma 3, every search, insertion or deletion in $BT(\mathcal{U})$ and in $BT(\mathcal{F})$ is done in $O(\log n)$ time. Therefore the construction of $Del(S)$ is done in $O(n \log n)$ time. Moreover, using the duality between $Vor(S)$ and $Del(S)$, $Vor(S)$ can be deduced in $O(n)$ time. □

8. Discussion

A question is : is it possible to reduce our algorithm to a sweep line algorithm ? This can not be done with an inversion of pole I since it transforms parallel lines into circles tangent at I.

Let φ be the mapping

$$\varphi : (r,\theta) \rightarrow (x',y') = \left(-r, \frac{2r\sin\theta}{1-\cos\theta}\right)$$

where (r,θ) are the polar coordinates of a point s of E and (x',y') the Cartesian coordinates of $\varphi(s)$. φ transforms the circles centered in O in a set of parallel straight lines but does not transform $Del(S)$ in $Del(\varphi(S))$.

We do not know of a better transformation that verifies this property.

Practical experiments show that the implementation of this algorithm is slower (~20%) than the sweep line algorithm. In our algorithm, the size of the front is bigger. This can be explained by the fact that the number of vertices of the convex hull of a set of n sites uniformly distributed in a circle is in $O(\sqrt[3]{n})$ but this number is in $O(\log n)$ for sites uniformly distributed in a rectangle.

However, the size of the binary tree of the ultimate points can be reduced by about a half. Only the locally minimal ultimate points, i.e., those whose distance to O is less than that of their two neighbors in $\mathcal{F}$, are inserted in the tree.

The relative execution times of the different steps of the algorithm are :

- ultimate point and ultimate site events : 38 %
- site events : 47 %
- sorting and memory management : 11 %
- updating of the map : 4 %

9. Conclusion

We have given a new way to compute the Delaunay diagram of a set S of n sites using an increasing circle in $O(n \log n)$ time. This algorithm is slower than the sweep line algorithm but it is practical if we have to build the Delaunay diagram only locally around a given point.

The sweep circle method has also been used to construct convex hulls and farthest point Delaunay diagrams in the plane. In these cases a shrinking circle is used and the algorithms generally end without sweeping all the sites [1, 2].

References

[1] B. Adam, P. Kauffmann, D. Schmitt, J.-C. Spehner; *Sweep algorithms for planar convex hulls*, Technical Report, Laboratoire MAGE, Université de Mulhouse (1996).

[2] B. Adam, P. Kauffmann, D. Schmitt, J.-C. Spehner, *A shrinking-circle algorithm for the planar farthest site Delaunay diagram*, Technical Report, Laboratoire MAGE, Université de Mulhouse (1996).

[3] F. Aurenhammer, *Voronoi diagrams- A survey of a fundamental geometric data structure*, A.C.M. Computational Surveys, 23, 3, (1992), 345-405.

[4] D. Beauquier, J. Berstel, Ph. Chrétienne, *Eléments d'algorithmique*, Masson, (1992), 411-425.

[5] J.-D. Boissonnat, M. Yvinec, *Géométrie Algorithmique*, Ediscience International (1995).

[6] J. Edmonds, *A combinatorial representation for polyhedral surfaces*, Notices Amer. Math. Soc. 7, (1960), 646.

[7] S. Fortune, *A sweepline algorithm for Voronoi diagrams*, Algorithmica 2, (1987), 153-174.

[8] P. Kauffmann, J.-C. Spehner, *Sur l'algorithme de Fortune*, Revue internationnale de CFAO et d'informatique graphique, Volume 10 - n°4/1995, 321-336

[9] R. Seidel, *Constrained Delaunay triangulations and Voronoi diagrams with obstacles*, in Report 260 Graz, Austria, (1988), 178-191.

Index of Authors